Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

Luís Miguel Marques Silva

LIMBus: Lightweight monitoring system powered by iOS and BLE

LIMBus: Sistema de monitorização com iOS e BLE

Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

Luís Miguel Marques Silva

LIMBus: Lightweight monitoring system powered by iOS and BLE

LIMBus: Sistema de monitorização com iOS e BLE

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Maria Fernandes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Ilídio Castro Oliveira (co-orientador), Professor Auxiliar do Departamento de Engenharia de Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Tomás António Mendes Oliveira e Silva
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Pedro Miguel Alves Brandão
Professor Auxiliar da Faculdade de Ciências da Universidade do Porto

Professor Doutor José Maria Amaral Fernandes
Professor Auxiliar da Universidade de Aveiro

agradecimentos /
acknowledgements

Agradeço toda a ajuda a todos os meus colegas e companheiros, assim como ao meu orientador e co-orientador, sem quem o resultado deste trabalho não teria sido possível. Agradeço também à minha família e à minha namorada pelo seu apoio incondicional.

Palavras Chave

bluetooth low energy, message queueing, mqtt, ios, monitorização.

Resumo

O advento da Internet das Coisas (IoT) tem levado a uma utilização mais intensa das redes de sensores que incluem tipicamente um nó agregador capaz de (1) comunicar com sensores próximos e (2) encaminhar os dados obtidos para um servidor de retaguarda, recorrendo habitualmente a tecnologias de transporte comuns da Internet. Os smartphones, com uma utilização popular, proporcionam um agente conveniente para implementar o papel de agregador e porta de saída (gateway) dos dados. Quando se estuda as soluções existentes neste contexto, verifica-se que a plataforma Android é a mais utilizada, com pouca expressão das soluções baseadas em iOS, apesar da disponibilidade de dispositivos. O desafio neste trabalho é avaliar a utilização da plataforma iOS para suportar dispositivos com o papel de agregador local e gateway, num cenário de IoT. Importa notar que houve avanços recentes na plataforma iOS, sendo agora possível ligar a sensores externos com Bluetooth Low Energy, bem como suportar o tratamento de fluxos contínuos de transmissão (streams). No entanto, ainda não é bem conhecido qual o impacto destas capacidades no desenvolvimento de soluções de redes de sensores e da sua aplicação em cenários reais. Neste contexto, desenvolvemos o sistema LIMBus, uma solução completa (do sensor à visualização remota) que explora a utilização da plataforma iOS, incluindo a integração de sensores com Bluetooth Low Energy e o envio dos dados para um sistema central, com visualização na Web. O LIMBus utiliza dispositivos com iOS para se ligar a sensores externos por Bluetooth Low Energy, fazendo um uso extenso dos perfis normalizados do Bluetooth Low Energy, e envia os dados dos sensores para o sistema central recorrendo a um protocolo por mensagens. No LIMBus é possível em tempo (quase) real visualizar os dados provenientes dos sensores através de uma aplicação Web. O LIMbus foi demonstrado em dois cenários diferentes: um cenário na área da saúde, com a monitorização básica de parâmetros fisiológicos, e um cenário na área do combate aos fogos, em que uma câmara térmica, desenvolvida na própria Unidade de investigação, é usada para detetar possíveis focos de calor. O desenvolvimento do LIMbus permitiu verificar que o novo suporte do iOS ao Bluetooth Low Energy e aos perfis Bluetooth Low Energy permite uma integração rápida com sensores (capazes de Bluetooth Low Energy) que, aliado a uma solução de transporte orientada por mensagens, proporciona uma plataforma viável e alternativa ao Android para cenários de IoT.

Keywords bluetooth low energy, message queueing, mqtt, ios, monitorization.

Abstract

Sensor networks are starting to be widespread with the advent of Internet of Things, typically relying on an aggregator that is able to: (1) efficiently communicate with nearby sensors and (2) relay the data to the consumer backoffice usually using Internet friendly protocols. Smartphones, given their pervasiveness, offer a convenient agent for the role of sensor data gateway. However, when surveying the existing smartphone based solutions Android is the most used, with little expression from iOS based solutions despite their availability. The challenge of this work is to assess iOS as a data aggregator and gateway solution in a IoT scenario. Recent advances extended iOS and brought new standardized integration with Bluetooth Low Energy sensors and support for sensor streaming handling, but it still remains to be seen what is the impact in realistic scenarios and if it allows iOS to be a valuable option. In this context, we propose an end-to-end system, LIMBus, that relies on iOS to integrate a Bluetooth Low Energy based sensor and provides a data gateway to a web connected backoffice. LIMBus uses iOS based handhelds to connect to sensors using Bluetooth Low Energy technology with extended use of Bluetooth Low Energy standard profiles, and relays the sensors data stream to a remote backoffice using a message based protocol. Using LIMBus it is possible to review in near real-time data from the sensors or persisted on the backoffice database. LIMBus was demonstrated in two different scenarios: health domain scenario integrating basic physiological monitoring, and a first responder scenario for heat sources localization, using a Bluetooth Low Energy compliant thermal sensor integrated in an in-house solution. LIMBus has illustrated that iOS new Bluetooth Low Energy support and Bluetooth Low Energy profiles allows a quick integration with Bluetooth Low Energy compliant sensors and together with a messaging based solution can provide a valid option to Android based system in IoT scenario.

Contents

List of Figures

List of Tables

Acronyms

ACL	Access Control List
ADP	Apple Developer Program
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BLE	Bluetooth Low Energy
BR/EDR	Bit Rate/Enhanced Data Rate
CoAP	Constrained Application Protocol
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DDS	Data Distribution Service
ECG	Electrocardiogram
GATT	Generic ATTribute Profile
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
MFi	Made for iPhone/iPod/iPad
MOM	Message Oriented Middleware
MQTT	Message Queueing Telemetry Transport
NDA	Non Disclosure Agreement
ODB-II	On-Board Diagnostics II
P2P	Peer-to-peer
QoS	Quality of Service
RF	Radio Frequency
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SDK	Software Development Kit
SoC	System-on-Chip
STOMP	Simple Text Oriented Messaging Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UUID	Universally Unique Identifier
UART	Universal Asynchronous Receiver/Transmitter
XMPP	Extensible Messaging and Presence Protocol

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The general availability of internet access and the ubiquitous use of smart personal devices, namely smartphones, raise the feasibility of new mobile applications, targeting connected scenarios.

This global communication system enabled the development of remote monitoring solutions in a cost effective way. Scientists can monitor the environment of remote locations without the need for a physical presence, allowing a better measurement and understanding of the natural phenomena. By remote monitoring their operations, industries can optimize them, decreasing costs and increasing performance.

People can make decisions about their lives based on information that is produced remotely by their smart appliances too. For example, with smart refrigerators, people can be notified if they are running low on some product, allowing them to make a small detour on their way home to buy it.

But maybe more important is the use of these new monitoring systems for health applications, the so called mHealth (mobile health) [1]. mHealth has the potential to dramatically reduce healthcare costs, either by prevention [2] or by giving healthcare professionals the means to remotely monitor and give feedback to patients [3], thus reducing the number of visits by the patient to the hospital or clinic.

BLE [4] is a recent technology from the Bluetooth SIG that was designed with low cost and energy efficiency in mind and allows devices to communicate within a distance of a few meters, from months to years [5], using a small button cell battery, making it suitable for this emerging market.

The use of mobile devices as gateways for data collected by sensors has been widely acknowledged, including in the health domain [6] [7] [8] [9]. For this purpose, the usage of Android based solutions with Bluetooth Bit Rate/Enhanced Data Rate (BR/EDR) (version < 4.0) (referred sometimes as Classic Bluetooth) is higher, than the usage of iOS with BLE. Although no comparative survey could be found, the search results in the literature support this difference. Further, the iOS based solutions usually require some sort of interaction from the user for the data to be uploaded (or that the application be running in foreground), due to the backgrounding restrictions present in iOS [10] [11].

The substantial market penetration of iPhones, and iOS in general, favours the opportunity to use these as gateways to provide the necessary connectivity for sensors data to be streamed to the

cloud. The evolution of Bluetooth in iOS, with support of BLE, and the introduction of background functionality for applications suggests that some of the hurdles for iOS adoption in sensor systems may have been overcome. This dissertation will assess if the new BLE iOS framework and background modes provide suitable support for using the iOS system as both a sensor data aggregator and as a data gateway in sensor networks.

1.2 OBJECTIVES

There is a large market penetration of iOS devices that support BLE, with some estimates pointing to 94% of the iPhones currently being used and 71% of the iPads [12]. This presents an opportunity to explore these devices as energy-efficient and BLE compliant data-gathering gateways. With this in mind, the LIMBus system, leverages the BLE capabilities, which allows an iPhone or iPad to act as a gateway for data being acquired by nearby sensors, to be remotely visualized.

The main objectives of this work are:

- Explore BLE profiles instead of traditional Peer-to-peer (P2P) Bluetooth BR/EDR socket stream and explore the new Application Programming Interface (API) in iOS
- Develop an iOS based gateway solution to relay data acquisition from BLE compliant sensors to web clients
- Use a decoupled solution between iOS gateway and sensor data consumers
- Illustrate the system in two realistic scenarios with different types of sensors: a health monitoring scenario and a temperature monitoring solution

1.3 CONTRIBUTIONS

The main contributions of LIMBus (Lightweight iOS Monitoring middleware) are:

- LIMBusCore – An iOS library that manages the BLE profile based connections with sensors and data acquisition.
- LIMBusTransmission – An iOS library that relays the data to online clients on the web. The solution relies on a messaging broker to manage the data exchange between LIMBusTransmission and the client – decoupling the library from transport related concerns.
- A BLE compliant interface for e-Health kit used in the health scenario – although not the main focus, this result implied a "low level" integration from the existing sensing solution with a BLE board, the nRF51-DK [13] – resulting in the port of the original e-Health library [14] to the mbed platform.
- LIMBus for iOS – An iOS application, gateway for BLE compliant sensors.
- LIMBusWeb – A web client for visualization of the relayed sensor data.
- Thermal Camera – a component for the decoding and visualization of temperature data from the thermal camera sensor.

- A publication in Mobiquitous – LIMBus: a lightweight remote monitoring system powered by iOS and BLE [15]

1.4 STRUCTURE

This dissertation is comprised of seven chapters:

- **Chapter 1:** Introduction – This chapter describes the context of this dissertation, its objectives and contributions.
- **Chapter 2:** State of the Art – This chapter reviews the iOS platform, the Bluetooth Low Energy technology and Message Oriented Middleware.
- **Chapter 3:** Scenarios – This chapter describes the scenarios which where used to test and validate this work and their requisites.
- **Chapter 4:** The BLE Health Kit Integration – This chapter describes the work done to integrate the e-Health Kit sensors into a Bluetooth Low Energy compliant sensor.
- **Chapter 5:** System Architecture and Implementation – This chapter presents the system architecture and describes its implementation.
- **Chapter 6:** System Validation – This chapter presents results of the system.
- **Chapter 7:** Conclusion – This chapter discusses the results, reviews the work done and suggests some future work.

CHAPTER 2

State of the Art

This chapter presents a review of the Bluetooth Low Energy technology and the iOS platform, taking into consideration the objectives of this work. Although messaging is not an objective of this work, it influenced some decisions on the architecture. Because of this, a revision of Message Oriented Middleware is also included.

2.1 Bluetooth Low Energy

Development of what would become the Bluetooth Low Energy technology was started in 2001 by researchers at Nokia [16]. The objective was to develop a technology, based on the already established Bluetooth technology, that would consume less power and be less expensive [17]. In 2006 the technology became public under the name Wibree [17] and in 2007, after negotiations with the Bluetooth SIG, an agreement was reached to include Wibree on the Bluetooth specification under the name Bluetooth Low Energy, the new specification (version 4.0) was completed in 2010 [18].

In the new specification, a BLE device can be single-mode or dual-mode. A single-mode device can only communicate with other BLE devices. A dual-mode device can communicate with both Bluetooth Low Energy and Bluetooth BR/EDR devices. This distinction is needed because the Low Energy and BR/EDR are not compatible, due in part to the changes made to the Physical Layer and packet format in order to reduce power consumption. Still, part of the hardware stack can be shared (e.g., the radio and antenna).

2.1.1 Specification Overview

The definition of how two devices exchange data is done by Generic ATTribute Profile (GATT) and GATT transactions are, in turn, defined by high-level concepts: Profiles, Services, Characteristics and Descriptors.

A profile defines a structure of elements such as services and characteristics used to fulfill a certain use case. The profile itself does not appear on a BLE device, only its composing elements. The Bluetooth SIG has adopted several standard profiles in the BLE specification – and continues to adopt more (e.g., Automation IO, Proximity Profile 1.1) – that describe and define common uses of the technology. Such standard profiles include support for devices that measure the heart rate, the blood glucose level or the weight, and can be accessed and visualized on the Bluetooth Developer Portal [19]. It is with these profiles that manufacturers of hardware and developers of software can ensure the technical compatibility of their products with others.

Each component of a profile is identified using a Universally Unique Identifier (UUID) and has a relationship as represented in figure 2.1:

- Service: a meaningful collection of characteristics and/or relationships to other services used to accomplish a certain function.
- Characteristic: is a value with properties and information about how it can be accessed and how the value can be interpreted/displayed.
- Descriptor: attribute that can be included in a characteristic to help describe a characteristic value, i.e., how it can be interpreted/displayed.

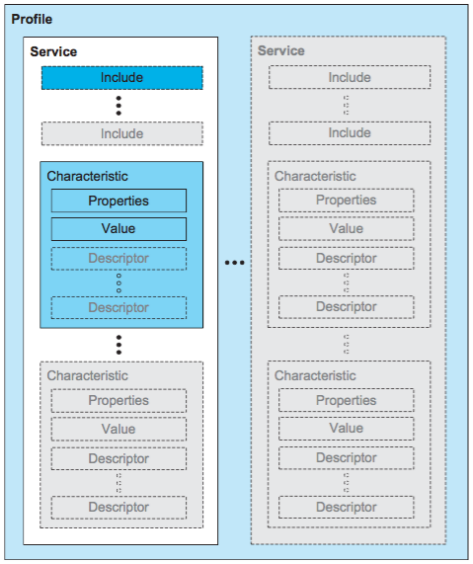


Figure 2.1: GATT-based Profile Hierarchy [4]

2.1.2 BLUETOOTH LOW ENERGY APPLICATIONS AND SENSORS

There are numerous applications of the BLE technology ranging from consumer electronics to industrial. The Bluetooth SIG alone has adopted several profiles (as seen in Table 2.1) and each company can develop its profiles as needed.

In health and fitness we find not only activity trackers as the Fitbit [21] or heart rate monitors as the Wahoo Blue Heart Rate Monitor [22], but also research initiatives trying to take it to another level by developing an Electrocardiogram (ECG) monitor [23].

ANP	Alert Notification Profile
AIOP	Automation IO Profile
BLP	Blood Pressure Profile
CGMP	Continuous Glucose Monitoring Profile
CPP	Cycling Power Profile
CSCP	Cycling Speed and Cadence Profile
ESP	Environmental Sensing Profile
FMP	Find Me Profile
GLP	Glucose Profile
HOGP	HID over GATT Profile
HRP	Heart Rate Profile
HTP	Health Thermometer Profile
IPSP	Internet Protocol Support Profile
LNP	Location and Navigation Profile
OTP	Object Transfer Profile
PASP	Phone Alert Status Profile
PXP	Proximity Profile
PLXP	Pulse Oximeter Profile
RSCP	Running Speed and Cadence Profile
ScPP	Scan Parameters Profile
TIP	Time Profile
WSP	Weight Scale Profile

Table 2.1: BLE Adopted Profiles [20]

There are however other less obvious applications that show the range of use cases this technology has; a study on the BLE potential for vehicular applications developed a small example for a keyless system [24]. A keyless system is used to unlock a vehicle whenever the owner is near, something that is usually achieved using Radio Frequency (RF) technology. By using the BLE technology, the energy consumption in the key can be reduced, while maintaining the usability of the system.

2.2 IOS

iOS is Apple's operating system that currently runs on the iPhone / iPad / iPod. It was introduced in 2007 with the launch of the iPhone [25], albeit at the time it had not yet been named iOS. This operating system has its roots on Apple Mac OS X [25] [26], and it is Darwin based and Unix-like. Only after the 2nd major release (version 2.0), and the introduction of the App Store, did it gain support for third party applications to be installed by the general public [27].
For third party applications to be created, Apple made available a public version of the iOS Software Development Kit (SDK) [28] and the Integrated Development Environment (IDE) Xcode [29]. The SDK contains all the frameworks and libraries needed to create an application. Xcode comes with a collection of tools which enable developers to debug, test and profile the applications they are creating

[30].

Up until recently, developers that were not enrolled in the Apple Developer Program (ADP) could only test their applications on the iOS Simulator, a tool that comes with Xcode. To deploy and test the applications on a physical device (iPhone/iPad/iPod) an enrolment in the ADP was required, which incurred in costs – this membership did however also include the option to publish the application in the App Store and access to the latest versions of iOS (betas) [31].

Since Jun 2015 [32], Apple merged all its ADP's (iOS, OS X, watchOS, Safari and, more recently, tvOS) into one single program and, with the new program, anyone can develop and test applications on a physical device, only requiring the paid membership to publish applications in the App Store [33].

When the iPhone 4S was released, it was the first smartphone to be shipped with support for the new and fresh BLE [34] and, with it, a new freedom came to iOS and developers of hardware, either hobbyists or professional. Previously, the communication with external accessories, either via Bluetooth BR/EDR or the "iDevice" (iPhone/iPad/iPod) connector, was limited to those developed under the Made for iPhone/iPod/iPad (MFi) program [35], a licensing program for developers of hardware and software of accessories [36]. This program requires a special request to Apple and most of it is under a Non Disclosure Agreement (NDA) [35].

With the advent of BLE, the CoreBluetooth framework [37] was introduced (iOS 5 [38]) and, with it, support for the development of applications that interact with BLE devices. This framework provides access to the BLE hardware in an iOS device and the functionality for applications to discover and interact with BLE devices.

To develop BLE devices, it is no longer required to be a member of the MFi program [35], so anyone can develop accessories that communicate with iOS applications using BLE. Initially the CoreBluetooth framework provided support for applications to communicate with BLE enabled devices (designated peripherals, in BLE terminology), such as heart rate monitors, thermometers, weight scales, etc. Since iOS 6, this framework also allows that an application acts as a peripheral [39], producing data for other devices to consume [40]. Currently, the CoreBluetooth framework allows an application to assume the central role (client), enabling it to discover, explore and interact with peripheral devices (servers, those who produce data) or to assume the peripheral role (producing data for a central device to consume) [41].

The mix of BLE with mobiles exploded the "appcessories" (accessories with a companion application) world. Fitbit [21] is a well known "appcessory" that makes use of the BLE technology in iOS. It keeps track of the steps taken and overall activity of a person and has a companion application that connects to it, using the BLE technology, and displays the data gathered by the Fitbit in a visually rich form, enabling a better understanding of it by the user. This application also allows this data to be saved to the cloud, making it accessible to the user anywhere [42].

Another example of this sensor cloud connectivity through an iPhone is the Automatic Adapter [43], it connects to the car central computer, using the On-Board Diagnostics II (ODB-II) port, and using BLE sends data from the car computer to the iPhone, which uploads it to the cloud. This enables a user to better understand fuel consumption, track their car and, also, discover possible problems with the car.

The backgrounding capabilities (i.e., the functionality when the application is not in the foreground) of iOS applications had always been restricted until iOS 4. In iOS 4, three background modes were introduced: Audio, VoIP and Location. With these modes, applications could play media while in the

background (e.g., music players), receive and sustain voice over IP phone calls or receive GPS location updates.

In iOS 5, and with the introduction of the CoreBluetooth framework, three new background modes were introduced: Newsstand Content, External Accessory and Bluetooth Central [44]. Of these three, the Bluetooth Central mode is the most important given the nature of this work. This mode added the ability for applications to communicate with BLE devices while in the background. Meaning that an application can discover, explorer and push or pull data from a BLE device, while on background. There are some limitations to these communications while the application on background but they are mostly related to priority and the speed what which the BLE scans are performed [45].

The open support of BLE gives iOS developers the chance to test new ideas with hardware devices without having to make a considerable financial commitment. In a world where more and more devices are gaining connectivity, this gives the developers and iOS a chance to play a significant part in the Internet of Things (IoT) revolution.

2.3 MESSAGE ORIENTED MIDDLEWARE AND MQTT

From the start, using a messaging solution was a basic requirement on the system to ensure an easy integration with existing solutions used in the group [46]. For the communication of distributed applications, traditionally developers had to design and implement custom protocols that would be used over TCP/IP. This is a time consuming task and prone to errors.

With the creation of Remote Procedure Calls (RPCs) and, sub-sequentially, other similar systems, as Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM) and Java Remote Method Invocation (RMI), a higher level of abstraction was introduced which allows applications to perform operations across different system platforms by invoking a local procedure or method. The logic to perform the remote call is abstracted from the developer and happens "behind the scenes". As with traditional calls to procedures or methods, in these systems, the caller of a procedure or method is required to block and wait until a response is obtained and, as such, to achieve parallelism, multi-threading must be used, which adds to the complexity of the applications and dramatically increases the number of possible problems. Another requirement is that to use RPCs, both the client and the server need to be online simultaneously, which in many cases might not be a desirable constraint when developing the applications, since it requires more logic to handle those edge cases and introduces more complexity and possible errors.

Message Oriented Middleware (MOM), by being decoupled and asynchronous by nature, provides a solution to the problems described above. MOM creates an abstraction of a message queue over the network and is generalization of the "the mailbox" operating system construct.

MOM provides a decoupled layer which allows applications to communicate by sending messages to each other using a central actor (broker) and decouples dependencies on programming data structures needed in the procedure-call approaches. There is no requirement for applications to be simultaneously online, the broker will store the message and deliver it when possible. Historically speaking most MOM implementations were closed source, but recently some open source protocols and implementations were

introduced: Advanced Message Queuing Protocol (AMQP), Data Distribution Service (DDS), Simple Text Oriented Messaging Protocol (STOMP), Extensible Messaging and Presence Protocol (XMPP), Message Queueing Telemetry Transport (MQTT) (Table 2.2).

	AMQP	DDS	MQTT	STOMP	XMPP
QoS	Delivery, Bandwidth, Errors	+20 parameters	At most once, At least once, Exactly once	Broker Reception Acknowledge	No[1]
Constrained Resources	No	No	Yes	No	No
Patterns	Publish/Subscribe, Request/Reply, Point-to-point	Publish/Subscribe	Publish/Subscribe	Publish/-Subscribe	Publish/-Subscribe[2]
Exchange Types	Fanout, Direct, Topic	Fanout	Topic	Topic	
Scalability	Dependent of implementation	Implementation requirements well defined in the specification	-	-	-

Table 2.2: MOM Comparison

The AMQP is an open standard that started being developed around 2003, it is designed to support different programming environments and operating systems by having a hard set of rules and behaviours of how it is expected to work [48]. It supports a number of different messaging patterns: asynchronous directed messaging, request/reply, publish/subscribe, store and forward [49].

The DDS is a Machine-to-Machine (M2M) publish/subscribe standard that is highly scalable and flexible [50].

STOMP is a text based publish/subscribe protocol. It was designed with simplicity and interoperability in mind and has a frame based protocol modelled on Hypertext Transfer Protocol (HTTP) [51].

XMPP is communication protocol based on XML mainly developed for instant messaging and presence [52]. It has extensions that give it publish/subscribe functionality [47].

[1] Only through the use of extensions.
[2] Using the 'XEP-0060: Publish-Subscribe' [47] extension.

MQTT is an open source, lightweight and simple, Client Server publish/subscribe messaging protocol [53].

Considering that our focus is in iOS, a major concern was on iOS support of the messaging solution. After a survey and some technical trials, MQTT showed to be the better option due to the very good support of the MQTT protocol provided by the MQTTKit library [54].

In MQTT protocol, as in other message queueing protocols, a message broker is needed. This broker receives messages published by clients and distributes it to other clients that have subscribed to them. To identify where the messages are to be sent and whom should receive them the concept of topic is used. A topic is a simple string much like a directory path in any *nix system, e.g., /this/is/a/mqtt/topic, /house/room/sensor. This organization allows a client to subscribe to the root of a topic (i.e., /house) and receive messages sent to all its subtopics.

MQTT has a Quality of Service (QoS) feature with 3 levels in increasing complexity, bandwidth and processor usage:

- QoS 0 – At most once delivery: no quality of service is assured by MQTT; the messages will be transmitted in a best effort mode in which the transport layer Transmission Control Protocol (TCP) is the only assurance.
- QoS 1 – At least once delivery: the message is assured to be delivered at least once, there can be duplicated messages.
- QoS 3 – Exactly once delivery: the message is assured to be delivered exactly once.

CHAPTER 3

Scenarios

For requirements elicitation and validation of the work, two scenarios were selected: a health scenario and a firefighters scenario. In the health scenario, a subject's biometric data is continuously read and streamed to a server, allowing the remote monitoring of the subject physiological signals. The thermal camera scenario consists of an infrared sensor camera to be used by firefighters to analyse the ground temperature and help determine if a fire is extinct.

3.1 Health scenario

The health scenario is a physiological monitoring scenario. In this scenario, a subject heart rate and temperature is remotely monitored using a pulsioximeter attached to a subject's finger and a thermometer placed in between the thumb and the index finger. For this scenario the e-Health kit [55] was chosen, which supplies multiple sensors including the pulsioximeter and thermometer. This scenario tries to simulate a use case where a patient has permission to leave the hospital (e.g., someone who had a heart problem) but is required to keep track of certain biometrics signals, allowing these signals to be gathered automatically and periodically.

The heart rate and temperature values read from the sensors are sent to the smartphone application. The application displays those values and uploads them to the server so that they can be visualized remotely.

3.1.1 e-health module

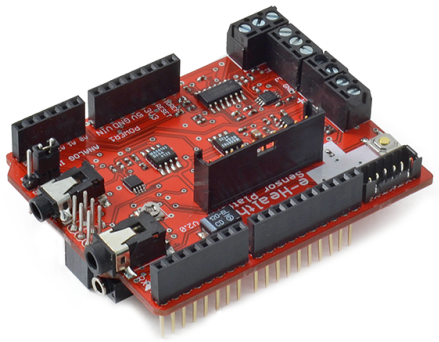


Figure 3.1: e-Health Shield [55]

The e-Health shield (part of the e-Health Kit) is a board (figure 3.1) that has inputs for several sensors, such as: thermometer (figure 3.2a), pulsioximeter (figure 3.2b), electrocardiogram (ECG) (figure 3.2c), Galvanic Skin Response Sensor (GSR) (figure 3.2d), and more. This board then exposes the sensor data using a pin connector that is compatible with the Arduino boards (and can also be used with Raspberry Pi by using the companion adapter).

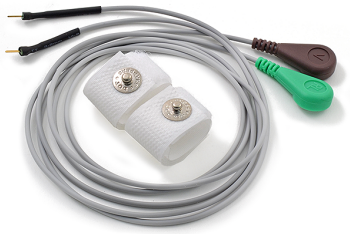
An "off-the-shelf" sensor-device with Bluetooth Low Energy capabilities could have been used, but given its high cost and given that there was already available an Arduino/Raspberry Pi sensor kit that could be adapted and used for this purpose, the already available Cooking Hacks e-Health sensors module was chosen.

The e-Health module is a hub which gathers several sensors terminals and provides an uniform access to their values. This module is coupled with the nRF51-DK, which provides the necessary programming and BLE functionality.

(a) Thermometer [55]

(b) Pulsioximeter [55]

(c) ECG [55]



(d) GSR [55]

(e) e-Health Shield with Sensors [55]

Figure 3.2: e-Health Sensors

E-HEALTH LIBRARY

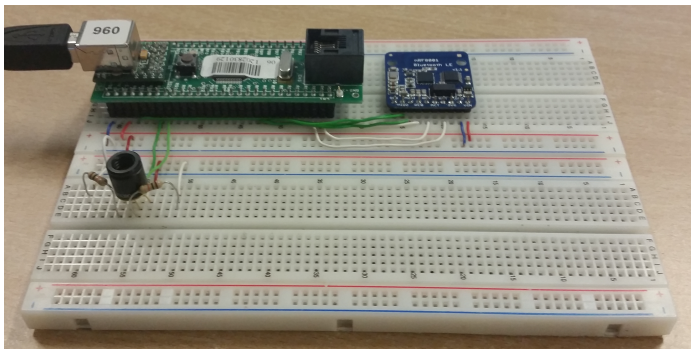
The e-Health module comes with a library that converts its physical sensors electric signals into a more usable information, e.g., temperature in Celsius or heart rate in BPM. This library does all the calculations necessary to convert the electric signals coming from the sensors terminals into meaningful and easily accessible data.

3.2 FIREFIGHTERS SCENARIO

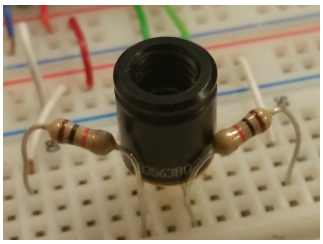
This scenario builds on the work being developed in the context of the VitalResponder project [56]. In this scenario, large amounts of data (in the BLE context) are streamed in real-time from the sensor to the smartphone application. It consists in using a thermal sensor device, being developed, as a way for firefighters to analyse the ground of a burned area while searching for hidden fire spots. After a fire is extinct, usually there are spots where some biological material (e.g., tree roots) keeps burning under the ground. To terminate these burning spots, the firefighters do a visual inspection of the ground, looking for signals of fire. The goal of this thermal camera is to enable firefighters to see, in a deterministic way (i.e., view the temperature in Celsius degrees), the ground temperature and determine more accurately if it is a fire spot.

3.2.1 THERMAL CAMERA

The thermal camera sensor (Figure 3.3a) was developed in-house in a M.Sc. dissertation. It is currently in the prototype stage and has no ready-to-use library available. It has a sensor (Figure 3.3b) with 64 photocells, each produces a value with a range between -50,00 ºC and 300,00 ºC. To transmit this value, it first converts it into an integer. Multiplying it by 100, thus removing the floating-point decimal component, the number is transformed into a 16-bit integer which can be sent using a BLE characteristic.



(a) Thermal Camera Prototype Board



(b) Infrared Sensor

3.2.2 THERMAL CAMERA AND BLE

To increase the efficiency of the transmissions, values from multiple photocells are grouped and sent over the same characteristic. These groups consist of 10 values because of limitations of the hardware, which can only send 20 bytes per characteristic in each transmission. 7 characteristics, with consecutive UUIDs, are used to transmit this data (see Table 3.1). The characteristics values are read periodically (200 milliseconds); after being read the value of each photocell is decoded and displayed in a 64 squares matrix (4x16). Each of these squares is coloured according to the temperature value it represents. Using a range of colours following the HUE format, going from dark blue (coldest) to hot red (hottest).

UUID	Photocells
E7880001-5E76-40E8-B49D-23F21B69A9F2	0 - 9
E7880002-5E76-40E8-B49D-23F21B69A9F2	10 - 19
E7880003-5E76-40E8-B49D-23F21B69A9F2	20 - 29
E7880004-5E76-40E8-B49D-23F21B69A9F2	30 - 39
E7880005-5E76-40E8-B49D-23F21B69A9F2	40 - 49
E7880006-5E76-40E8-B49D-23F21B69A9F2	50 - 59
E7880007-5E76-40E8-B49D-23F21B69A9F2	60 - 63

Table 3.1: Thermal Camera Characteristics

CHAPTER 4

The BLE health kit integration

4.1 INTRODUCTION

Enabling the e-Health kit with BLE capabilities was necessary to fulfill the proposed health scenario. Although not the main focus of this thesis, it implied a non trivial and time consuming integration of the e-Health hardware with a BLE hardware interface. Such interface does not exist (as used in our solution) and can be considered a side relevant contribution of the system; this result is referred as LIMBusEmbedded.

To create a BLE profile based compliant sensor for the health kit, we needed to integrate it with a board that provided such functionality. The nRF51-DK has a chip with an integrated BLE modem, the so called System-on-Chip (SoC), and provided a good choice. It has a pin format that makes it directly compatible with the health kit module (as seen in Figure 4.1) and is supported in the mbed platform [57]. The nRF51-DK makes possible the exposure of the health kit data through a BLE interface and, the mbed platform makes it easier to create such interface by providing a high level abstraction of the microcontroller and modem of the nRF51-DK. The e-Health kit module is connected to the nRF51-DK, which enables the data coming from sensors terminals to be passed onto the nRF51-DK and using the e-Health library is decoded so that it can be processed and exposed using the correct BLE profiles.

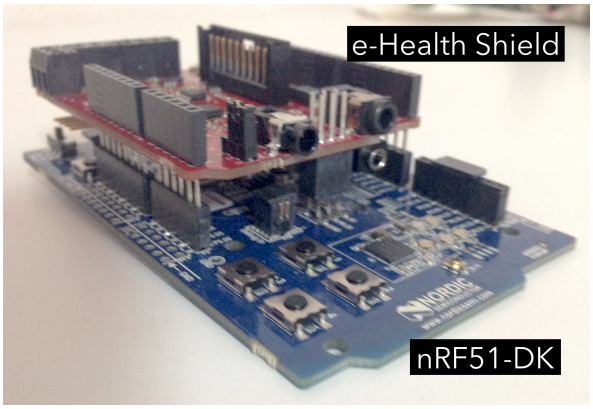


Figure 4.1: nRF51-DK coupled with e-Health Shield

4.2 nRF51-DK

The nRF51-DK was chosen because of its very good BLE capabilities and because its board format is similar to that of an Arduino, which enables it to be coupled with Arduino compatible modules, as is the case of the e-Health module.

This board, being mbed enabled, brings to the table a great advantage: it enables rapid prototyping by taking care of the burden of implementing all the low level microcontroller details, much like the Arduino.

Unlike the Arduino platform, mbed gives full access to the boards hardware while still providing an easier entry-point for development of embedded software. In this case in particular there was the need to create a device with a Heart Rate Profile and Health Thermometer Profile, something that would not be possible with the Arduino platform because all its the BLE modules are "too" abstracted and have a certain way of being used, not allowing the creation of services or characteristics, either standard or custom. All the modules found come with a preconfigured Universal Asynchronous Receiver/Transmitter (UART) style service (a characteristic to write and a characteristic to read) that is used as a virtual socket to exchange packets. Though this would be sufficient to create a sensor device that would expose data via BLE, it would not comply with the standard way of exposing the data established by the Bluetooth SIG and the device would not be generic, used only for the purpose of this work.

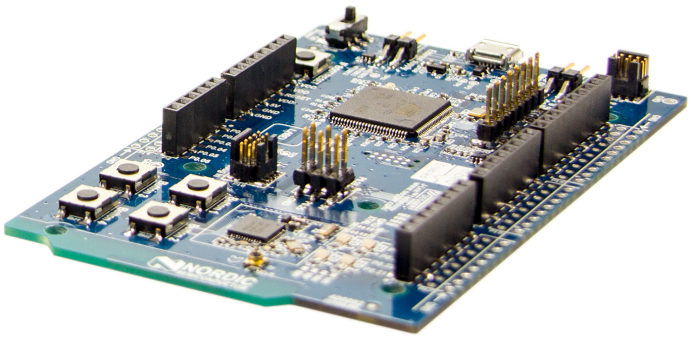


Figure 4.2: nRF51-DK [13]

4.3 MBED

The ARM mbed is a platform for IoT that provides cloud services, tools and an ecosystem for developers to create IoT devices. It can be thought of as Arduino on steroids. It is programmed using C++ and has an online IDE (figure 4.3) with integrated version control which allows people to work in the same project more easily than Arduino.

The online IDE takes care of compiling and generating the program binary file. When a mbed board is connected to a computer it mounts itself as a flash pen. To program the board a simple drag and drop operation of the downloaded program binary file into this pen starts the flashing process.

mbed provides high level abstractions over the compatible boards while also allowing direct access to its hardware. In this aspect, Arduino seems to make it harder to access the underlying hardware.

A great aspect of this platform is that the boards use modern chips which work with 3 and even 1.8 Volts (versus the Arduino 5 Volts old microcontrollers), which enables the development of really low energy devices. This platform also allows a smoother transition from prototype to production than the Arduino platform, since the generated program binary files are ready to flash, which means that there is no requirement for the chip to be previously provisioned with a boot-loader. On top of that, the boards provide access to a JTAG port which enables step by step debugging.

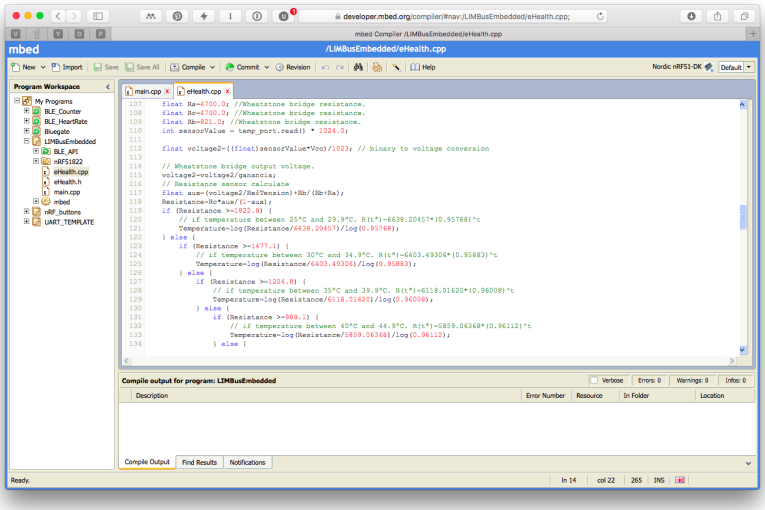


Figure 4.3: mbed online IDE with the LIMBusEmbedded project

4.4 LIMBUSEMBEDDED

4.4.1 OVERVIEW

Using the mbed platform and its BLE library (BLE_API), the device exposes 3 services (Heart Rate Service, Health Thermometer Service, Device Information Service).

An instance of BLEDevice is used to control the BLE functionalities of the the device and this object is responsible for all BLE operations.

To create the services, an instance of HeartRateService, HealthThermometerService and DeviceInformationService, respectively, is created. These instances are then passed to the BLEDevice before it starts advertising its services.

The program then enters an infinite loop in which the values of the heart rate and temperature sensors are read and published. The temperature value is read from the sensor using the e-Health library method getTemperature(), which returns the temperature in Celsius.

The Temperature Measurement characteristic is then updated using the updateTemperature(float) method in the HealthThermometerService object instance. The heart rate measurement works a bit differently. The eHealth library continuously (with a certain frequency) reads the heart rate values from the pulsioximeter and stores them in a public variable named BPM. This program then reads the value present in that variable and, using the method updateHeartRate(int) of the HeartRateService object instance, updates the Heart Rate Measurement characteristic.

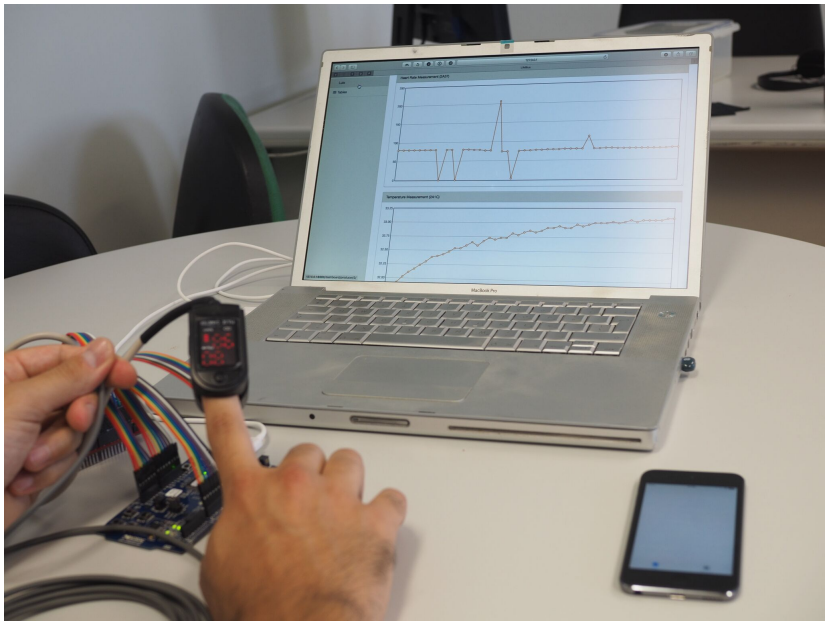


Figure 4.4: LIMBus hardware in use setup

4.4.2 e-health library

Although the e-Health module can be easily attached to the nRF51-DK, its signal processing library only has support for Arduino or Raspberry Pi, there is no mbed compatible library. Because of that, a partial port of the library had to be made before the e-Health module could be used. The port consisted mainly of correcting the board pins mapping, but also the readjustment of the frequency at which the pulsioximeter sensor is read from. It is suspected that this readjustment was needed because of differences between the Arduino boards and the nRF51-DK.

It was also detected that some noise was introduced into the readings from the pulsioximeter, because of this a simple filter was added. This filter works by ignoring the BPM values for which the magnitude of change from the last value is greater than 10. Although not perfect this filter eliminated most of the noise coming from the sensor.

CHAPTER 5

System architecture and implementation

5.1 System architecture

The objective of this dissertation was the exploratory use of the iOS platform and BLE technology, to implement a sensor's data aggregator and gateway, with a decoupled architecture, that enables the remote monitoring using BLE enabled sensors. The proposed solution is the LIMBus system.

The main components of LIMBus are (1) a mobile data aggregator, (2) the MQTT broker, (3) backend services and (4) a web application (figure 5.1). The mobile data aggregator is an application running on an iOS device; it collects data from sensors wirelessly, using BLE, and relays the incoming data to the backend server, which runs a MQTT messaging broker. The data aggregator uses the BLE protocol features to discover nearby sensors. The backend services, running on a remote server, receives the incoming data via the MQTT broker. The broker notifies the registered LIMBusWorker process whenever it receives new sensor data and LIMBusWorker stores the received data in a database for long term storage. The web application accesses this long-term storage and provides a structured, and in almost real-time, visualization of the data coming from one or more sensors.

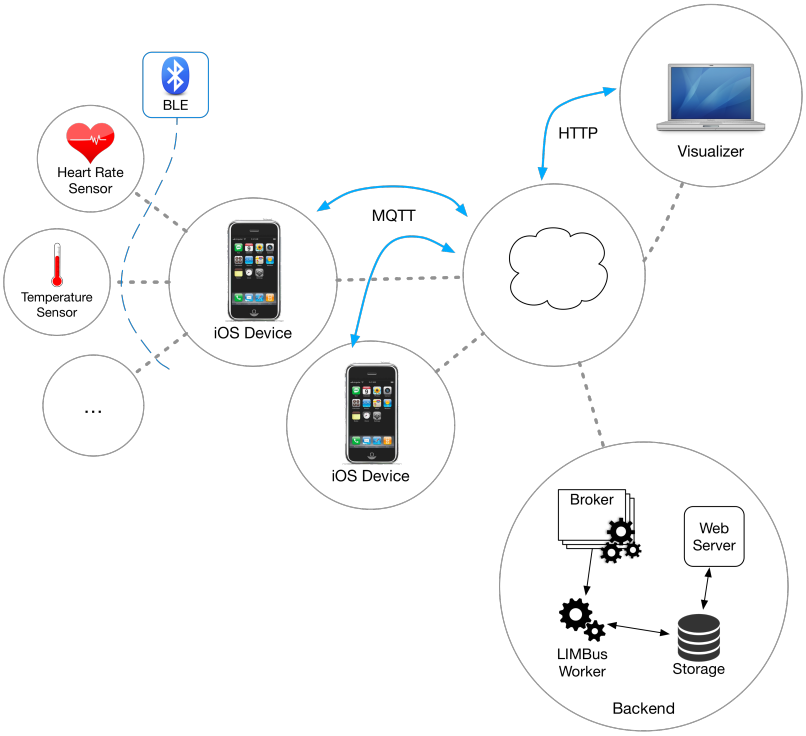


Figure 5.1: LIMBus Architecture – Components interconnection and organization

5.1.1 COMPONENTS INTERACTION

This architecture targets typical remote online monitoring use cases. The iOS application (LIMBus for iOS), acting as a daemon process, initializes the CoreBluetooth framework as described per the Apple documentation [58].

As can be seen in figure 5.2, it starts scanning for BLE nearby peripherals. When a peripheral is found, a connection is established to discover which services and characteristics it provides. If it finds characteristics of interest (identified by a UUID that has been previously provisioned), it starts reading and relaying the characteristic data to the remote MQTT broker. The broker analyses the received data by the topic it was written to and, after decoding it, saves it in the database, using the model represented in figure 5.3. This data is then access by the LIMBus web application, which decodes it from the BLE format and displays it.

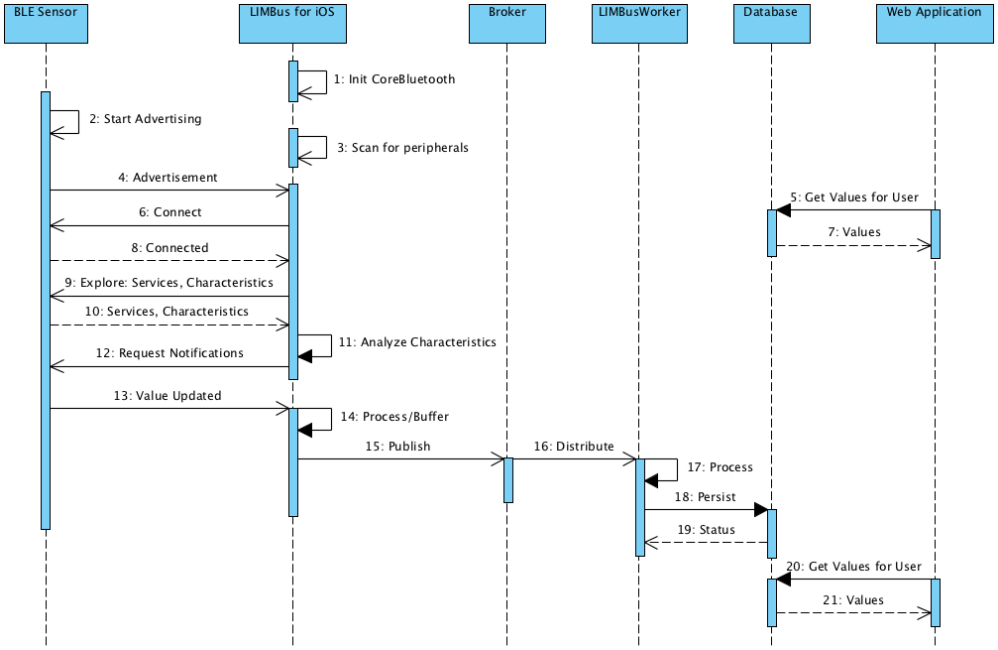


Figure 5.2: System Interaction Overview

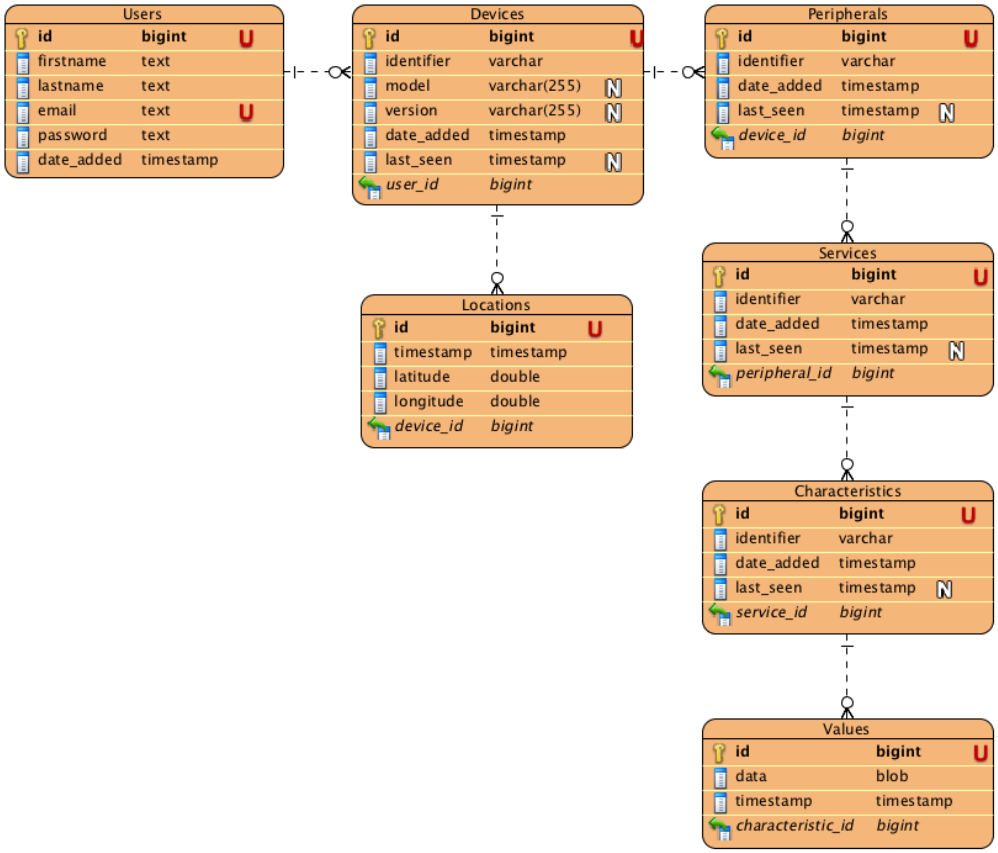


Figure 5.3: Server Database Model – Used to store users and BLE information, and sensors data

5.1.2 SENSORS TO AGGREGATOR BLE STREAMS

In the BLE data model, as seen in figure 2.1, each peripheral can have several services which, in turn, can have several characteristics. For instance, our test device (peripheral) provides two data streams: a heart rate monitor and a thermometer that, in BLE, are represented as two different services. In the specific case of the heart rate service, it allows retrieving the current heart rate measurement and the minimum and maximum within a time interval. These could be represented by 3 individual characteristics associated with the service. Each individual service and characteristic has its own unique identifier (UUID) associated. The service and characteristic UUIDs in BLE allow anyone to infer from any BLE peripheral the data semantics without knowing its inner-workings, based on standard BLE profiles.

5.1.3 AGGREGATOR TO BACKEND MESSAGING

LIMBus makes use of a MQTT broker for message passing – in our implementation we use the Mosquito MQTT broker [59]. The broker controls what can be read/written using @<>@<>@acls (ACLs) [60], so that an unauthorized user cannot access data belonging to another user. Different user contexts are mapped to different topics (e.g. "/users/<username>"). All communications for a specific user are done via subtopics. LIMBusWorker on the backend connects to the MQTT broker and subscribes to all the users subtopics using the wildcard '#' (e.g. subscribing to "/users/#" allows to receive all messages published to subtopics of "/users"), to receive the incoming data and store it in the database. The topic to where the message is published (listing 1) has the information about device, peripheral, service and characteristic. The characteristic values are encoded using @<>@<>@json (JSON) following the model in Figure 5.4 for which the value binary data is Base64 encoded.

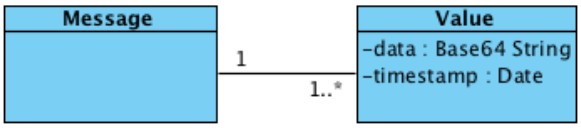


Figure 5.4: MQTT Characteristics Message Model

```
/users/<UserEmail>/data/<DeviceUUID>/<PeripheralUUID>/<ServiceUUID>/<
    CharacteristicUUID>
```

Listing 1: MQTT Topic Format

COAP

Although a messaging solution was the preferred choice, another protocol was analysed at the start of this work, mainly due to its potential: @<>@<>@coap (CoAP) [61]. CoAP follows a RESTful approach [62] and uses @<>@<>@udp (UDP) as the underlying transport protocol [61]. As MQTT, it

provides low overhead and fast transmission, being good at transmitting data using as little bandwidth as possible.

Given its RESTful approach and interoperability capabilities with HTTP servers, CoAP was a good contender since the server API could be developed using standard RESTful methods and then by only installing a CoAP proxy the requests and responses would be translated between the application and the server [61].

The support in iOS was, however, lacking. The only library that exists for this protocol is iCoAP iCoAP [63] and it has a very naïve implementation, seriously lacking functionality and incapable of transmitting binary data – only strings.

DETAILS OF LIMBUSWORKER

LIMBusWorker, being the application that is responsible for persisting in the server the data being streamed from LIMBus for iOS, needs to have read-access to all the topics into which the values are being published. To accomplish this, LIMBusWorker identifies itself with the username 'master' (this was previously provisioned in the passwords file of Mosquitto) which gives it special privileges to access all the subtopics of '/users' (as described in listing 2). To receive the messages, and data, published into the subtopics of '/users', it subscribes the topic '/users/#'. In this case the wildcard '#' means all subtopics of '/users' and subsequent hierarchy.

The messages topics have the format described in Listing 1.

LIMBusWorker splits and extracts the user email and UUIDs, using this information to know where to store the characteristic values and creating the necessary entries in the corresponding database tables and also the necessary relationships, as can be seen in figure 5.5.

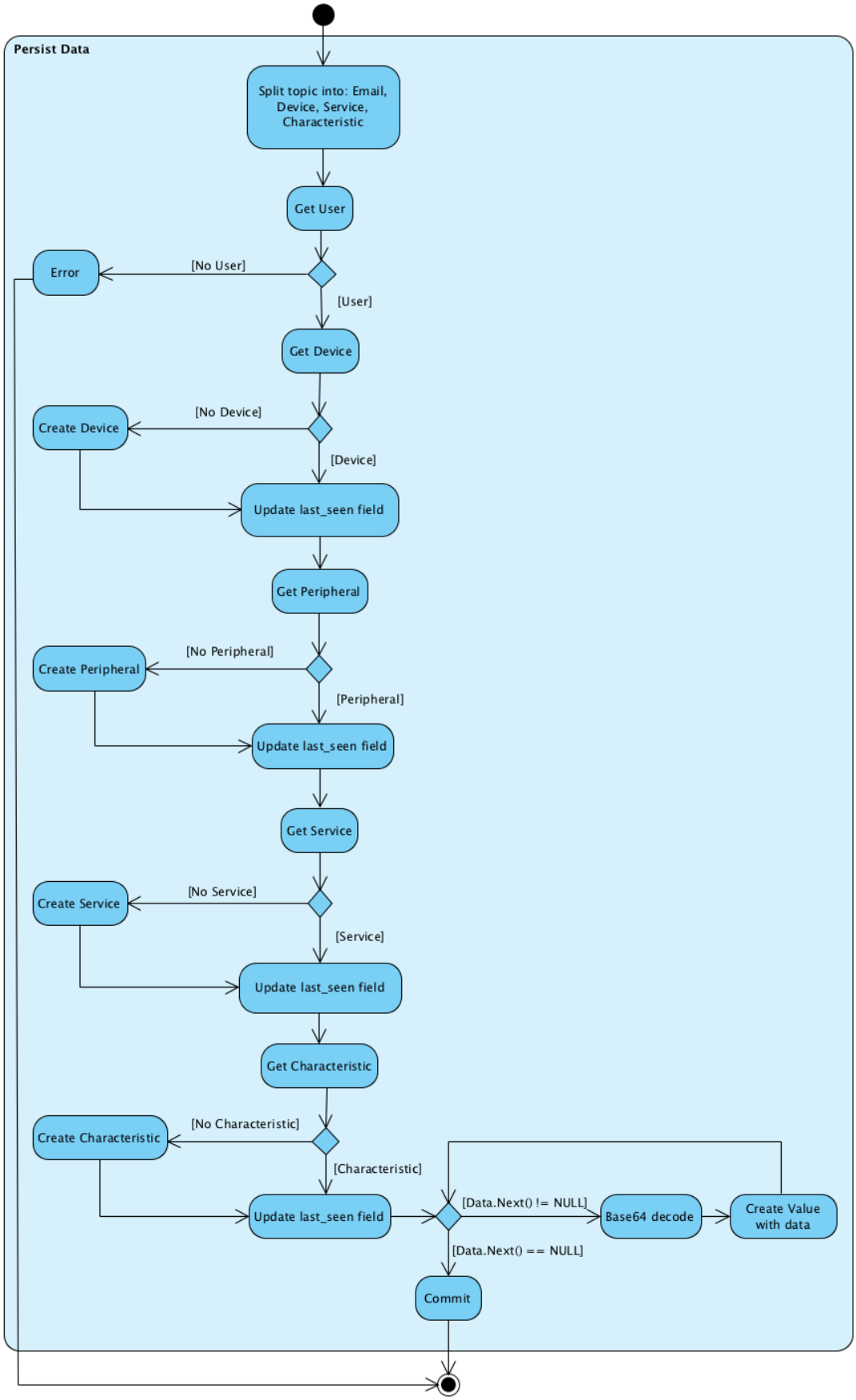


Figure 5.5: LIMBusWorker sensor data storage activity diagram

5.2 IMPLEMENTATION

The implementation of this system was organized in 5 main parts: (1) heart rate and temperature sensor that exposed its information using @<>@<>@ble, (2) a mobile application developed for iOS, (3) a MQTT broker and messages processor, (4) a database to persist the information coming from sensors, (5) a web application to visualize the information coming from sensors in real-time.

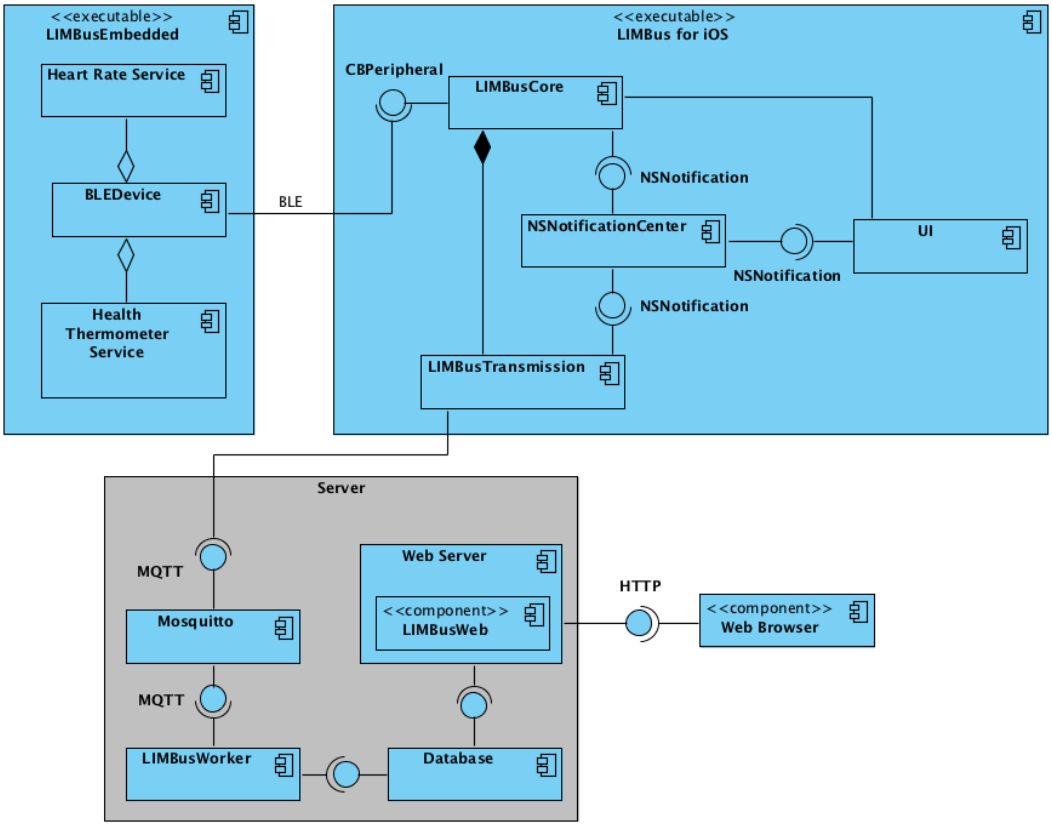


Figure 5.6: LIMBus System Components Diagram

5.2.1 MOBILE APPLICATION

This application is responsible for reading data from nearby known BLE sensors, process it and periodically upload it to the server (even while on background).

The description of how this application was implemented is divided in 3 main areas: @<>@<>@ble operations and sensors communications, internet connection and server functionality, user interface presentation and thermal camera functionality.

BLUETOOTH LOW ENERGY FUNCTIONALITY

The BLE functionality is implemented in LIMBusCore. It handles the configuration necessary for the application to support BLE, scans for sensors, connects and manages connections to sensors, reads data either periodically or whenever the sensor has new data. It is also responsible for keeping and

maintaining the list of paired sensors. It has 2 modes of operation (regular, pairing), which can be switched on at any moment.

In the regular mode, after the initialization of the Bluetooth framework (CoreBluetooth), it starts to scan for sensors nearby, here-on referred to as peripherals. Whenever a peripheral is found, it's verified if the peripheral was previously paired and, if so, a connection attempt is made. If the attempt is unsuccessful, retries are made until it is established. After the connection has been established, the peripherals services are explored and sub sequentially its characteristics.

This component only reads values from characteristics that are classified as "of interest". What this means is that the component has a list of characteristic identifiers (UUIDs) for which to look for and read from. Whenever a characteristic "of interest" is found, its properties are analysed. If it has the notify property, this component subscribes to be notified by BLE when its value changes. If, on the other hand, it has the read property and does not have the notify property, it is inserted into a local list which is used by the application to know which characteristics values it needs to periodically read.

After the value of characteristic is read (either by being notified or by direct reading) an application-wide notification is sent, using NSNotificationCenter [64], that contains the value and information about the peripheral, service and characteristic to which the value belongs. The use of application-wide notification lessens the coupling of components, decreasing the complexity of the component and inherent problems to holding many references to other object instances that would like to receive the values updates. This decoupling allowed, for example, the easy integration of the thermal camera functionality.

In the pairing mode, this component only scans for peripherals and notifies their discovery to the calling object through the use of a call-back block – since this is a sporadic operation, the use of application-wide notifications felt like an overkill approach. This allows the easy pairing or un-pairing of peripherals. The list of paired peripherals is persisted using NSUserDefaults [65].

The reason for the introduction of characteristic "of interest", is that a peripheral can have services and characteristics which does not make much sense in reading within a certain context. For example, in the case of this work, although the peripheral exposes information about the device version, model, etc., it made little sense to continuously read those values and upload them to the server. The values that actually made sense to upload were the heart rate and/or temperature. So, this component would only look for those characteristics. This can, however, be configured.

The key 'bluetooth-central' was added to the background execution modes (UIBackgroundModes in Info.plist) for the application to receive values updates while in background and/or the mobile is in standby.

BACKGROUND OPERATION

While the application is in background it retains its BLE functionality by declaring the 'bluetooth-central' key in Info.plist UIBackgroundModes (as described above). This enables the application to be awaken whenever there is a BLE related event, being it the discovery of a new peripheral or the value update of a characteristic in a peripheral to which the application is connected.

All that is needed, for the application to have this capability, is the declaration of the key described above. When needed, the system awakens the application and calls the necessary delegate methods from CBCentralManagerDelegate or CBPeripheralDelegate. These are the same delegate methods used while the application is running in the foreground.

When the application is in background, it only works in regular mode (no pairing), connecting to

paired peripherals that are found, trying to re-connect whenever a connection is lost and reading the value of characteristics whenever a notification of value update is received.

After being awaken by a notification of a characteristic value update, the application reads its value and uploads it to the server, as it would if it was in foreground.

It should be noted, though, that this background operation only works properly if the peripheral notifies any connected device when its characteristics values are updated. This is due to the application only being awaken when there are BLE events (e.g., characteristic value updated notification). So, if a peripheral does not have that capability and only "waits" for its characteristics to be read, the application is not able to awaken itself automatically to read the values.

COMMUNICATIONS WITH SERVER

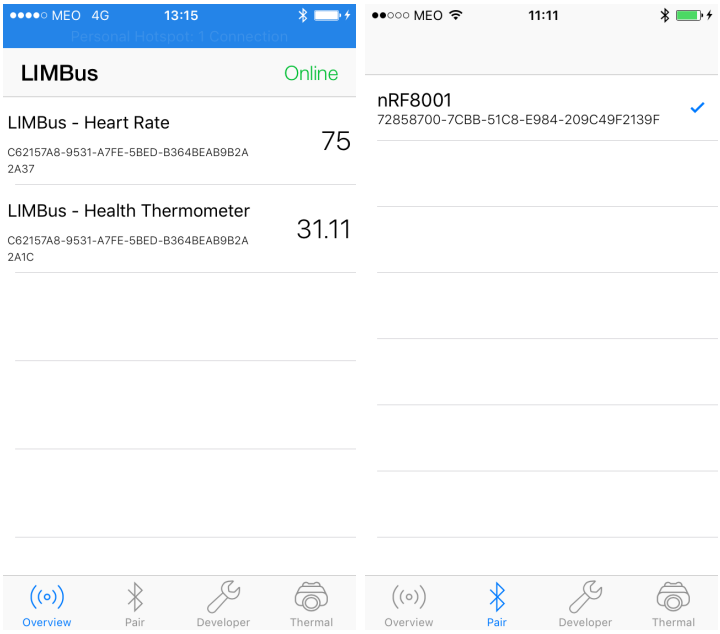
The internet communications functionality is implemented in LIMBusTransmission. Whenever the application receives a new value from a peripheral it sends it to this component which handles the upload of the data to the server. The value is accompanied of the device identifier, peripheral identifier, service identifier and characteristic identifier, which allows this component to publish the value into a topic with the format presented in listing 1. With this format all the necessary information about a value is present in the topic (which device aggregated and uploaded the value, belonging to which user, from which sensor).

The device identifier is an UUID that uniquely identifies the iPhone or iPad. This identifier is automatically generated by iOS and is the same between applications from the same vendor and while at least 1 application from the same vendor is installed. If all applications of a vendor are uninstalled this identifier will be different when an application is reinstalled. Applications from different vendors will have different device identifiers [66].

To improve the transmission efficiency, i.e., to try and upload the maximum amount of data in each transmission but without introducing significant latency, the values are buffered and sent periodically. This gives the component a chance to accumulate various values before performing the upload. The values are kept in a NSMutableArray stored inside a NSMutableDictionary and identified by a key which matches the format of the topic (and actually is the topic). Using the topic as key simplifies mapping the values on the dictionary to the published ones.

USER INTERFACE OF LIMBUS FOR IOS

The interface is currently divided in 4 screens. (1) Overview (5.7a), (2) Pair (5.7b), (3) Developer, (4) Thermal. The Thermal screen was added in a later stage, to allow the visualization of the temperature data coming from the thermal camera sensor, introduced more recently. In the Overview screen it is possible to see if there is a connection to the server and information about the characteristics currently being read from, if any. Figure (5.7a) shows that the application has connectivity to the server expressed by the green "Online" text. It also shows that the application is connected to the LIMBusEmbedded sensor (named "LIMBus") and is reading the values from the Heart Rate and Thermometer.



(a) Overview Screen (b) Pair Screen

Figure 5.7: LIMBus for iOS User Interface

The Pair screen shows peripherals nearby, allowing the user to pair or un-pair a peripheral. In Figure 5.7b we can see that the Thermal Camera is nearby and being detected by LIMBus for iOS and that it is already paired with the application.

The Developer screen displays the internal log; it is used mainly as a debug feature whenever the device is being used in the field.

The Thermal screen displays a matrix of 4 x 16 squares, each of these squares has a color corresponding to the temperature it is representing, going from cold blue to hot red (Figure 5.9). The screenshot seen in Figure (5.8) shows the values read from the camera when a hand was put in front of it.

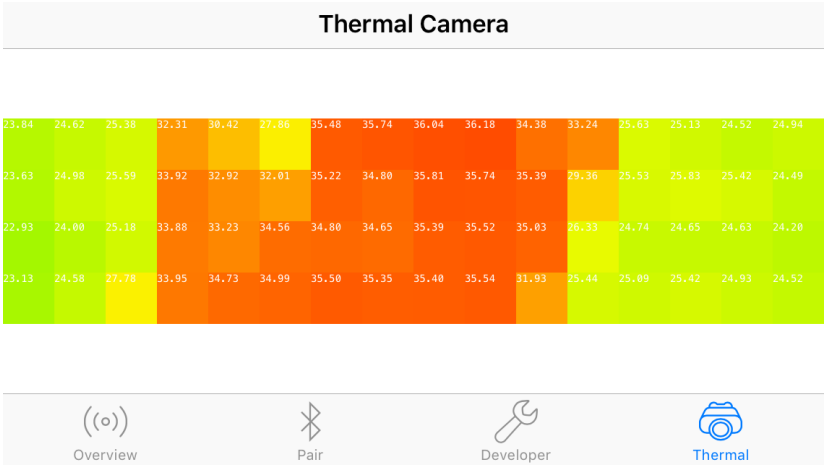


Figure 5.8: Thermal Camera Screen

Figure 5.9: Thermal Camera Temperature Gradient

5.2.2 MESSAGES BROKER AND LIMBUSWORKER

The MQTT broker deployed in the server is Mosquitto. Although not strictly required, given that the purpose of this work was not to analyse this broker authentication/authorization features, it was configured to only give read or write privileges to certain users. A regular user – a user that is not 'master' – can only write to a topic with its email and has no read privileges. With these constraints in place, a user cannot read data being published by another user and so, data leaks are blocked. The created ACL is presented below in listing 2, this ACL can be placed anywhere in the system, as long as its path is correctly entered in the broker configuration file using the option 'acl_file <path>'. The configuration file can itself be located anywhere in the system, an its path is passed as an argument into Mosquitto.

```
pattern write /users/%u/data/#

user master
topic read /users/#
topic write /users/#
```

Listing 2: Mosquitto Access Control List

The '%u' keyword, seen in Listing 2, is used to define the "current user", or in other words, the user that is publishing the message. The MQTT broker verifies if the topic to which the user is trying to publish the message matches the topic with the correct user email. An example of what would happen if the user 'lmsilva@ua.pt' would try to publish a message into a topic with his user email and into a topic with another user email is seen in 3.

```
User: lmsilva@ua.pt

PUBLISH -> /users/lmsilva@ua.pt/data - Authorized
PUBLISH -> /users/admin@ua.pt/data   - Unauthorized
```

Listing 3: Publish Authorization Example

5.2.3 LIMBUSWEB

A web application was developed to remotely visualize in real-time the data being produced by the sensors and uploaded by the iOS application.This web application was developed using the Django framework [67]. This is a high-level Python web framework that already provides many web functionalities allowing the faster development of web applications. In this framework a web page is rendered server-side and then sent to the browser.

Currently the web application supports (i.e., can decode the value of) the Heart Rate Measurement characteristic and the Temperature Measurement characteristic. The decoding is performed server-side, but could also be done on the client-side using JavaScript. The latter was initially tried, but because the characteristics values are stored as raw binary data, they had to be encoded into a Base64 string by the server before being sent to the client browser. The client browser would then need to decode the Base64 string into binary data and, on top of that, decode the characteristic value. This resulted in a lot of unnecessary overhead.

The information presented in LIMBusWeb is organized by user, with all the user associated characteristics being shown in a page. A graph is used to show a history of the characteristic values and the latest value is shown beside the graph. This information is refreshed periodically using AJAX by accessing a REST style endpoint developed for this purpose.

Figure 5.10 shows the heart rate (top) and temperature (bottom), of the user Luís. The temperature is lower than 37°C because the subject was holding the thermometer between the thumb and the index finger.

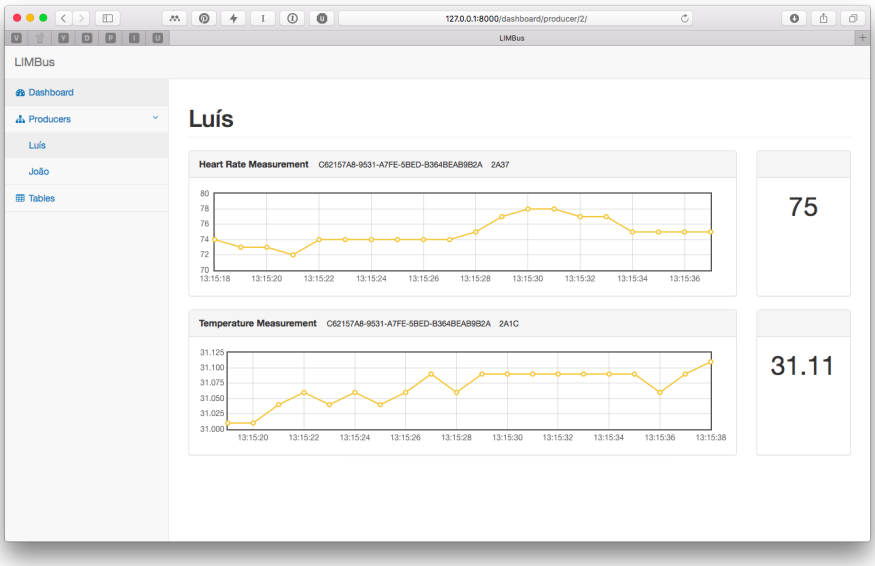


Figure 5.10: LIMBusWeb: Subject Heart Rate and Temperature

5.2.4 LESSONS LEARNED

IOS BACKGROUND EXECUTION AND BLE

While implementing this functionality, it was noticed that the application has access to the network stack and internet connectivity when awaken by the system to handle BLE related events. This allows the application to immediately upload the values coming from the BLE characteristics.

This is something that is not referred in the documentation, nor in any of the literature found, both in the academic and general world.

IOS AND BLE

Although not mentioned anywhere in the documentation, whenever a peripheral is discovered, the reference to its CBPeripheral instance must be retained (in LIMBusCore, an array is used to save all discovered peripherals). If this is not done, further operations with that peripheral, such as connect, have no result (an example project was created to demonstrate this: BLERetainTest), a snippet of the project is shown in Listing 4. The log output produced when the peripheral is not retained (as seen in the snippet) is shown in Listing 5, and as can be seen, the connect request as no effect. The log output after removing the comment is shown in Listing 6, and as can seen, the peripheral is correctly connected to.

The possible reason for this behaviour — and this is pure speculation — might be that the instance is not being automatically retained and as such is released after a while. So when the central manager receives the connection request result from the BLE device, it no longer holds the instance of the CBPeripheral for which the connection request was made in the first place, and cannot dispatch the results to the delegate.

```
var peripheralInstance : CBPeripheral!
func centralManager(central: CBCentralManager, didDiscoverPeripheral
    peripheral: CBPeripheral, advertisementData: [String : AnyObject],
     RSSI: NSNumber) {
  print("centralManager:didDiscoverPeripheral:␣\(peripheral.
      identifier.UUIDString)")

  centralManager.connectPeripheral(peripheral, options: nil)
  // peripheralInstance = peripheral
}

func centralManager(central: CBCentralManager, didConnectPeripheral
    peripheral: CBPeripheral) {
  print("centralManager:didConnectPeripheral:␣\(peripheral.identifier
      .UUIDString)")
  peripheral.delegate = self
  peripheral.discoverServices(nil)
}
```

Listing 4: Retain Test Code Snippet

```
centralManagerDidUpdateState: PoweredOn
centralManager:didDiscoverPeripheral: 85DDADE0-9151-C8C9-0345-75
    F93E61297B
```

Listing 5: Log: not retained

```
centralManagerDidUpdateState: PoweredOn
centralManager:didDiscoverPeripheral: 85DDADE0-9151-C8C9-0345-75
    F93E61297B
centralManager:didConnectPeripheral: 85DDADE0-9151-C8C9-0345-75
    F93E61297B
```

Listing 6: Log: retained

CHAPTER 6

System Validation

To validate this system, two tests were made: one with the health scenario and another with the thermal camera scenario.

For the health scenario the heart rate sensor was attached to a subject finger and the thermometer sensor was held between two fingers by the subject.

For the thermal camera scenario a finger was held in sight of the sensor and moved sideways.

6.1 background operation

Since its introduction, iOS has given applications a restricted functionality, if any, while they are in the background. Implementing and launching daemon processes as seen in general computing (or even in Android) is not possible in iOS. Over time, some functionality has been introduced into iOS that allows some tasks to be performed when an application is in the background, including BLE related tasks. Such functionality is described in the *Background Execution* section of the *App Programming Guide for iOS* [11].

To verify and validate the background operation functionality, a stripped version of the LIMBus for iOS application was created. This application contains only the necessary BLE functionality and provides tests for both MQTT connectivity and regular HTTP connectivity. With this stripped version a more objective and quantitative test can be conducted of the application background mode. Both this application and LIMBus for iOS interact in the same way with the CoreBluetooth framework and with MQTTKit, but in this new application it was introduced functionality for testing HTTP connectivity.

Below, the logs of the HTTP connectivity test are shown and since the results are equal to those of MQTT, the MQTT logs are omitted as it would add nothing to the results.

A special version of the LIMBusEmbedded firmware was also created. In this version, only the Heart Rate Profile is available and the heart rate measurement is incremented each second until it reaches the value 255, starting again from 0 when it does. Each time the heart rate measurement value is incremented, a notification is sent to any connected device. This provides a deterministic way of

analyzing the data received by the mobile application and verify if it's being correctly received in the server.

First the application was run without the background mode enabled. As can be seen in listing 7, as soon as the application goes to background (line 35), the value updates of the characteristic stop being received resuming only (line 36) when the application is brought to the foreground by the user almost 25 minutes later (line 37). This behavior is reflected on the server, which stops receiving the values as seen in listing 8.

```
1  14:43:23.131: application:didFinishLaunchingWithOptions:
2  14:43:23.295: applicationDidBecomeActive
3  14:43:23.564: centralManagerDidUpdateState: PoweredOn
4  14:43:23.729: centralManager:didDiscoverPeripheral: C62157A8-9531-
       A7FE-5BED-B364BEAB9B2A - BGTEST
5  14:43:24.751: centralManager:didConnectPeripheral: C62157A8-9531-A7FE
       -5BED-B364BEAB9B2A - BGTEST
6  14:43:25.242: peripheral:didDiscoverServices: C62157A8-9531-A7FE-5BED
       -B364BEAB9B2A - BGTEST
7  14:43:25.244: peripheral:didDiscoverServices -> Heart Rate
8  14:43:25.333: peripheral:didDiscoverCharacteristicsForService: Heart
       Rate
9  14:43:25.334: peripheral:didDiscoverCharacteristic: 2A37
10 14:43:26.061: Heart Rate Measurement: 8
11 14:43:27.048: Heart Rate Measurement: 9
12 14:43:28.069: Heart Rate Measurement: 10
13 14:43:29.058: Heart Rate Measurement: 11
14 14:43:30.048: Heart Rate Measurement: 12
15 14:43:31.068: Heart Rate Measurement: 13
16 14:43:32.058: Heart Rate Measurement: 14
17 14:43:33.047: Heart Rate Measurement: 15
18 14:43:34.068: Heart Rate Measurement: 16
19 14:43:35.058: Heart Rate Measurement: 17
20 14:43:36.047: Heart Rate Measurement: 18
21 14:43:37.068: Heart Rate Measurement: 19
22 14:43:38.059: Heart Rate Measurement: 20
23 14:43:39.048: Heart Rate Measurement: 21
24 14:43:40.068: Heart Rate Measurement: 22
25 14:43:41.059: Heart Rate Measurement: 23
26 14:43:42.048: Heart Rate Measurement: 24
27 14:43:43.068: Heart Rate Measurement: 25
28 14:43:44.058: Heart Rate Measurement: 26
29 14:43:45.047: Heart Rate Measurement: 27
30 14:43:46.068: Heart Rate Measurement: 28
31 14:43:47.058: Heart Rate Measurement: 29
32 14:43:48.048: Heart Rate Measurement: 30
33 14:43:49.068: Heart Rate Measurement: 31
```

```
34 14:43:49.259: applicationWillResignActive
35 14:43:50.016: applicationDidEnterBackground <- APPLICATION GOES TO
      BACKGROUND
36 15:07:47.173: Heart Rate Measurement: 189 <- APPLICATION COMES TO
      FOREGROUND
37 15:07:47.193: applicationWillEnterForeground
38 15:07:47.638: applicationDidBecomeActive
39 15:07:48.046: Heart Rate Measurement: 190
40 15:07:49.026: Heart Rate Measurement: 191
41 15:07:50.043: Heart Rate Measurement: 192
42 (...)
```

Listing 7: Application log: without background mode

```
1 (...)
2 14:43:46,146: GET /ble/28
3 14:43:47,144: GET /ble/29
4 14:43:48,135: GET /ble/30
5 14:43:49,150: GET /ble/31 <- APPLICATION GOES TO BACKGROUND
6 15:07:47,399: GET /ble/189 <- APPLICATION COMES TO FOREGROUND
7 15:07:48,156: GET /ble/190
8 15:07:49,116: GET /ble/191
9 (...)
```

Listing 8: Server HTTP log: without background mode

After this test, the background mode was enabled and the application run again. The application was kept in foreground for some time and then sent to background. As can be seen in listing 9, after going to background (line 19) the characteristic value updates keep being received, processed and sent to the server as can be seen in listing 10.

As can be seen in listing 10, the values might come out of order and the value 49 was never received. This failure is confirmed in the application log as seen in listing 11. Error -999 is described in the CFNetwork documentation [68] as "CFURLErrorCancelled: The connection was cancelled.", which is not very informative. It might be due to a timeout, since the error was only reported after a while.

```
1 15:08:47.635: application:didFinishLaunchingWithOptions:
2 15:08:47.797: applicationDidBecomeActive
3 15:08:48.068: centralManagerDidUpdateState: PoweredOn
4 15:08:49.071: centralManager:didDiscoverPeripheral: C62157A8-9531-
      A7FE-5BED-B364BEAB9B2A - BGTEST
5 15:08:50.103: centralManager:didConnectPeripheral: C62157A8-9531-A7FE
      -5BED-B364BEAB9B2A - BGTEST
```

```
15:08:50.512: peripheral:didDiscoverServices: C62157A8-9531-A7FE-5BED
    -B364BEAB9B2A - BGTEST
15:08:50.513: peripheral:didDiscoverServices -> Heart Rate
15:08:50.559: peripheral:didDiscoverCharacteristicsForService: Heart
    Rate
15:08:50.559: peripheral:didDiscoverCharacteristic: 2A37
15:08:51.044: Heart Rate Measurement: 216
15:08:52.031: Heart Rate Measurement: 217
15:08:53.021: Heart Rate Measurement: 218
15:08:54.040: Heart Rate Measurement: 219
(...)
15:10:15.040: Heart Rate Measurement: 44
15:10:16.031: Heart Rate Measurement: 45
15:10:16.881: applicationWillResignActive
15:10:17.021: Heart Rate Measurement: 46
15:10:17.623: applicationDidEnterBackground <- APPLICATION GOES TO
    BACKGROUND
15:10:18.856: Heart Rate Measurement: 47
15:10:19.547: Heart Rate Measurement: 48
15:10:20.027: Heart Rate Measurement: 49
15:10:21.047: Heart Rate Measurement: 50
15:10:22.037: Heart Rate Measurement: 51 <- VALUES KEEP COMING
(...)
```

Listing 9: Application log: with background mode

```
(...)
15:10:15,132: GET /ble/44
15:10:16,116: GET /ble/45
15:10:17,152: GET /ble/46 <- APPLICATION GOES TO BACKGROUND
15:10:19,581: GET /ble/47
15:10:21,538: GET /ble/50
15:10:21,681: GET /ble/48
15:10:22,571: GET /ble/51
(...)
```

Listing 10: Server HTTP log: with background mode

```
(...)
15:20:13.615: URLSession:task:didCompleteWithError: Error Domain=
    NSURLErrorDomain Code=-999 "(null)" UserInfo={NSErrorFailingURLKey
    =http://95.85.7.24/ble/49, NSErrorFailingURLStringKey=http
```

```
    ://95.85.7.24/ble/49, NSURLErrorBackgroundTaskCancelledReasonKey
    =0}
3 (...)
```

Listing 11: Application log: value upload failure

6.2 BLUETOOTH LOW ENERGY PERFORMANCE

The BLE transmissions are performed using a determined time interval, in the LIMBus system the interval used was 1 second because it was deemed sufficient for the physiological signals that were being analyzed.

A series of tests were conducted to analyze the transmission performance of BLE when the transmission interval is reduced. The tests used the same stripped version of LIMBus for iOS as above and the same adapted LIMBusEmbedded firmware as above.

To test different transmissions intervals the adapted firmware incremented the heart rate measurement before each transmission. The tested intervals were: 100 ms, 50 ms, 20 ms, 15 ms, 14 ms, 13 ms, 12 ms, 11 ms, 10 ms, 5 ms and 1 ms.

Tests performed while the application is in the background show very similar results to the tests performed while in foreground.

Period	Sent	Received	Lost	Loss %
100 ms	526	526	0	0.0
50 ms	574	574	0	0.0
20 ms	544	540	4	0,7
15 ms	570	566	4	0,7
14 ms	508	474	34	6,6
13 ms	519	444	75	14,4
12 ms	547	434	133	20,6
11 ms	651	472	179	27,4
10 ms	1879	1257	622	33,1
5 ms	4277	1449	2828	66,1
1 ms	5579	409	5170	92,6

Table 6.1: BLE Throughput Performance

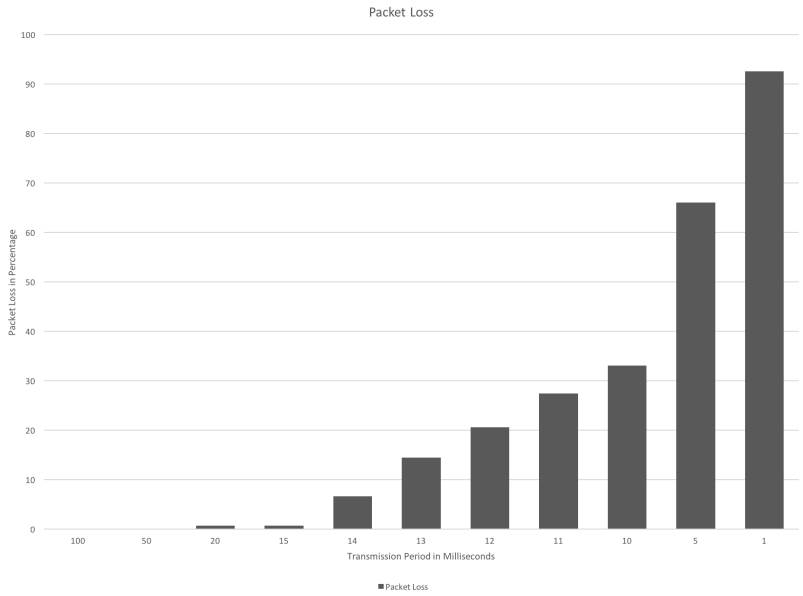


Figure 6.1: BLE Packet Loss

As can be seen in table 6.1, as the interval of transmission decreases, the packet losses start to increase. Noticeably, after the interval decreases below 15 milliseconds, between transmissions, the packet loss increases significantly.

6.3 HEALTH SCENARIO

As can be seen in the in Figure 6.2 the subject heart rate is 75 beats per minute and his temperature 30.96°C in LIMBusWeb and the temperature in LIMBus for iOS is 31.01°C, this is due to the small delay caused by communications between the iOS application, LIMBusWorker and web application. This progression can better be seen in the logs of LIMBus for iOS (12) and LIMBusWorker (13).

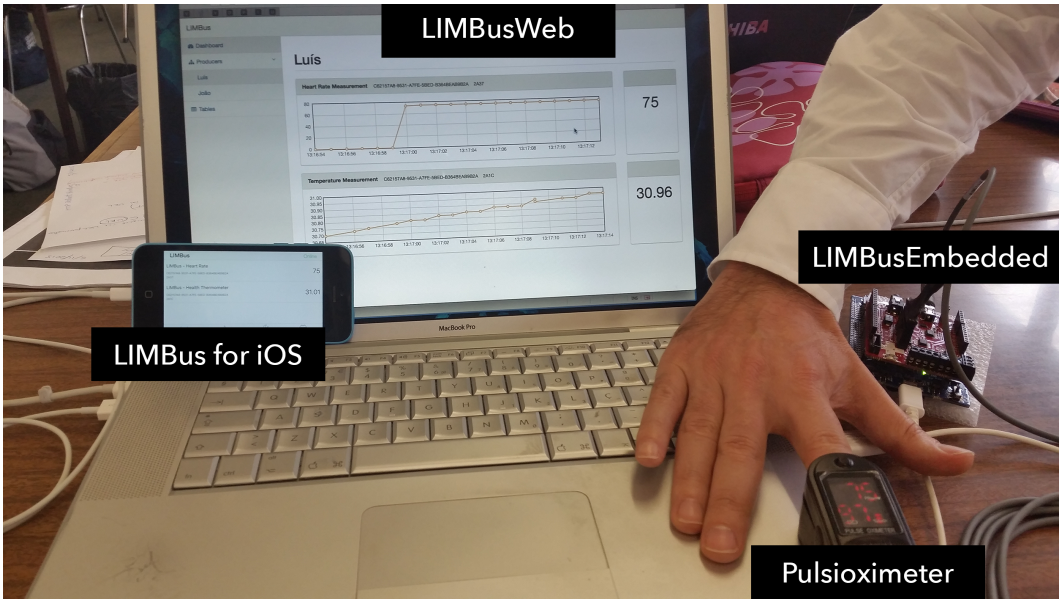


Figure 6.2: Health Scenario: subject being monitored

As can be seen in the log of LIMBus for iOS (12) the temperature data is received from the Temperature Measurement characteristic (UUID: 0x2A1C), at 13:17:14.041, and is buffered for transmission, at 13:17:14.057. Simultaneously an asynchronous application-wide notification, of the reception of this characteristic data, is sent. This notification is received in the 'Overview' view controller and its data is decoded, at 13:17:14.074, resulting in the LIMBusDecoder log output.

The same sequence happens to the heart rate data that arrives from the Heart Rate Measurement characteristic (UUID: 0x2A37).

```
1  (...)
2  13:17:14.041: LIMBusCore: Characteristic '2A1C' data -> <00180c00 fe>
3  13:17:14.057: LIMBusTransmission: encoded data for topic '/users/
       lmsilva@ua.pt/data/F4123D3E-870E-4A06-9475-38CCA7381BD6/C62157A8
       -9531-A7FE-5BED-B364BEAB9B2A/1809/2A1C' -> ABgMAP4=
4  13:17:14.074: LIMBusDecoder: Characteristic '2A1C' value -> 30.96
5  13:17:14.947: LIMBusCore: Characteristic '2A37' data -> <004b>
6  13:17:14.979: LIMBusDecoder: Characteristic '2A37' value -> 75
7  13:17:14.996: LIMBusTransmission: encoded data for topic '/users/
       lmsilva@ua.pt/data/F4123D3E-870E-4A06-9475-38CCA7381BD6/C62157A8
       -9531-A7FE-5BED-B364BEAB9B2A/180D/2A37' -> AEs=
8  (...)
9  13:17:17.029: LIMBusCore: Characteristic '2A1C' data -> <001d0c00 fe>
10 13:17:17.048: LIMBusTransmission: encoded data for topic '/users/
       lmsilva@ua.pt/data/F4123D3E-870E-4A06-9475-38CCA7381BD6/C62157A8
       -9531-A7FE-5BED-B364BEAB9B2A/1809/2A1C' -> AB0MAP4=
11 13:17:17.067: LIMBusDecoder: Characteristic '2A1C' value -> 31.01
12 13:17:17.946: LIMBusCore: Characteristic '2A37' data -> <004b>
```

```
13 13:17:17.980: LIMBusTransmission: encoded data for topic '/users/
       lmsilva@ua.pt/data/F4123D3E-870E-4A06-9475-38CCA7381BD6/C62157A8
       -9531-A7FE-5BED-B364BEAB9B2A/180D/2A37' -> AEs=
14 (...)
```

Listing 12: LIMBus for iOS log excerpt

On the server, the reception the temperature and heart rate data can be seen in the LIMBusWorker log excerpt (13). The buffering functionality of LIMBusTransmission can be seen in this log. On the reception of the heart rate measurement, 2 measurements are seen in the same message.

```
1  (...)
2  Received message in '/users/lmsilva@ua.pt/data/F4123D3E-870E-4A06
       -9475-38CCA7381BD6/C62157A8-9531-A7FE-5BED-B364BEAB9B2A/1809/2A1C'
       : [
3    {
4      "data" : "ABgMAP4=",
5      "timestamp" : "2015-11-05T13:17:13Z"
6    },
7    {
8      "data" : "ABgMAP4=",
9      "timestamp" : "2015-11-05T13:17:14Z"
10   }
11 ]
12 Received message in '/users/lmsilva@ua.pt/data/F4123D3E-870E-4A06
       -9475-38CCA7381BD6/C62157A8-9531-A7FE-5BED-B364BEAB9B2A/180D/2A37'
       : [
13   {
14     "data" : "AEs=",
15     "timestamp" : "2015-11-05T13:17:14Z"
16   }
17 ]
18 (...)
19 Received message in '/users/lmsilva@ua.pt/data/F4123D3E-870E-4A06
       -9475-38CCA7381BD6/C62157A8-9531-A7FE-5BED-B364BEAB9B2A/1809/2A1C'
       : [
20   {
21     "data" : "ABoMAP4=",
22     "timestamp" : "2015-11-05T13:17:15Z"
23   },
24   {
25     "data" : "AB0MAP4=",
26     "timestamp" : "2015-11-05T13:17:17Z"
27   }
28 ]
```

```
29 Received message in '/users/lmsilva@ua.pt/data/F4123D3E-870E-4A06
       -9475-38CCA7381BD6/C62157A8-9531-A7FE-5BED-B364BEAB9B2A/180D/2A37'
       : [
30   {
31     "data" : "AEs=",
32     "timestamp" : "2015-11-05T13:17:17Z"
33   }
34 ]
35 (...)
```

Listing 13: LIMBusWorker log excerpt

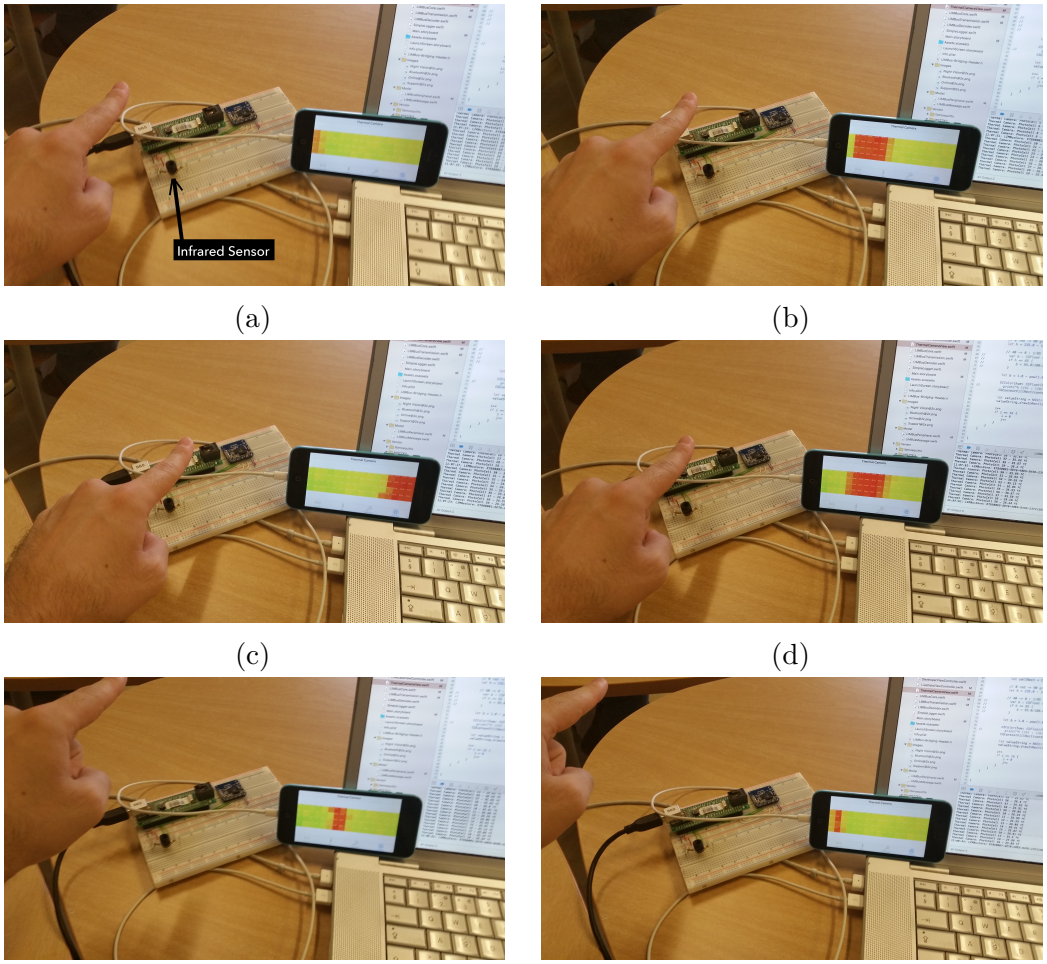
6.4 FIREFIGHTERS SCENARIO

To verify that the values sensed by the thermal camera were being correctly read and decoded by LIMBus for iOS, the temperature differential between a human finger and the ambient was analyzed.

In the sequence of images presented below, the finger is moved over the camera from left to right (6.3a -> 6.3b -> 6.3c), back to the center and away from the camera into to left (6.3d -> 6.3e -> 6.3f).

The thermal camera has a dynamic range of -50°C to 300°C, which would show little contrast when a human finger (36°C) is put in front of it against the ambient temperature (20°C).

So before collecting the images, the range of temperatures, in LIMBus for iOS, was reduced so that better contrast would be seen in the result. The mininum temperature was set to 10°C and the maximum to 50°C. The temperatures were color coded using a gradient as seen in figure 5.9.



(a) (b) (c) (d) (e) (f)

Figure 6.3: Thermal Camera Test

CHAPTER 7

Conclusion

The main objective of this dissertation was the exploratory use of iOS as a sensor data aggregator and gateway by leveraging its BLE functionalities and background execution capabilities.

The LIMBus for iOS application shows that iOS is a viable option when developing such systems and that iOS has come a long way since its early days in terms of application functionality restrictions. Not only can an iOS application continuously interact with BLE devices but also keep operating while in the background. Although not clear in the documentation, it was possible to learn that the application has access to the network stack while it is in the background, allowing it to communicate with remote servers when it is awaken to process BLE related events.

With the use of MQTT, for the communication between LIMBus for iOS and the remote server, the produced system maintains a level of decoupling which allows the easy integration into other existing systems. It also shows that the MQTT protocol is well supported in iOS, with MQTTKit leveraging the power of libmosquitto.

The health scenario shows that the system fulfils the proposed objectives, enabling a subject's biometric signals to be remotely monitored by using the LIMBus system. The use of mbed for the development of LIMBusEmbedded shows that it is a capable platform for rapid development of firmware; it allowed us to provide an in-house sensor device with standard BLE profiles. The integration of the eHealth kit resulted in a library that can be used in the mbed platform for interacting with the eHealth shield.

The firefighters scenario reveals that although the BLE technology was not made to transmit large quantities of data, it can address the problem requirements, proving to be a viable option, alternative to Bluetooth BR/EDR.

7.1 Future work

Further development of the LIMBusWeb application is required for real world experiments with subjects and health care professionals. Mainly, it would be required to implement support for authentication and authorization, and user interface improvement to have different functionality

available for different roles. Given the sensitive nature of such usage, measures to guarantee the data safety and privacy on the system and its communications also had to be put in place.

Although not directly related to this work, further analysis of the thermal camera sensor with in-field testing to verify its efficiency and discover possible optimizations in the communications between the sensor and LIMBus for iOS.

References

[1] R. Seewon, "New horizons for health through mobile technologies", vol. 3, ISSN: ISBN 978 92 4 156425 0. DOI: ISBN9789241564250.

[2] H. Larkin, "Mhealth", *Hospital & Health Networks*, vol. 85, no. 4, pp. 22–26, 2011, ISSN: 1068-8838.

[3] S. Agarwal and C. T. Lau, "Remote health monitoring using mobile phones and web services.", *Telemedicine journal and e-health : The official journal of the American Telemedicine Association*, vol. 16, no. 5, pp. 603–7, 2010, ISSN: 1556-3669. DOI: 10.1089/tmj.2009.0165. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20575728.

[4] Bluetooth Special Interest Group (SIG), "Bluetooth specification version 4.1", no. December, p. 2684, 2013.

[5] J. Liu, C. Chen, Y. Ma, and Y. Xu, "Energy analysis of device discovery for bluetooth low energy", *IEEE Vehicular Technology Conference*, pp. 1–5, 2013, ISSN: 15502252. DOI: 10.1109/VTCFall.2013.6692181.

[6] S. C. K. Lam, K. L. Wong, K. O. Wong, W. Wong, and W. H. Mow, "A smartphone-centric platform for personal health monitoring using wireless wearable biosensors", in *Information, Communications and Signal Processing, 2009. ICICS 2009. 7th International Conference on*, 2009, pp. 1–7, ISBN: 978-1-4244-4656-8. DOI: 10.1109/ICICS.2009.5397628.

[7] L. Zhong, M. Sinclair, and R. Bittner, "A phone-centered body sensor network platform: cost, energy efficiency & user interface", in *International Workshop on Wearable and Implantable Body Sensor Networks (BSN'06)*, 2006, pp. 179–182, ISBN: 0-7695-2547-4. DOI: 10.1109/BSN.2006.4. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1612925.

[8] C. Otto, A. Milenković, C. Sanders, and E. Jovanov, "System architecture of a wireless body area sensor network for ubiquitous health monitoring", *Journal of Mobile Multimedia*, vol. 1, no. 4, pp. 307–326, 2006, ISSN: 03050009. DOI: 10.1109/MC.2009.5. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.2522%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[9] M. Wagner, B. Kuch, C. Cabrera, P. Enoksson, and a. Sieber, "Android based body area network for the evaluation of medical parameters", *Intelligent Solutions in Embedded Systems (WISES), 2012 Proceedings of the Tenth Workshop on*, pp. 33–38, 2012.

[10] F. Zhang and H. Yu, "Aegeanboard: an interactive whiteboard messenger for ipad", *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pp. 257–258, 2014. DOI: 10.1109/MobileCloud.2014.42. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6834972.

[11] *Background execution – app programming guide for ios*. [Online]. Available: https://developer.apple.com/library/prerelease/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html (visited on 2015-10-15).

[12] *iOS version stats*. [Online]. Available: https://david-smith.org/iosversionstats/ (visited on 2015-10-24).

[13] *Nordic semiconductor nrf51-dk*. [Online]. Available: http://www.nordicsemi.com/eng/Products/nRF51-DK (visited on 2015-10-24).

[14] *Cooking hacks e-health library*. [Online]. Available: https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical#step3 (visited on 2015-10-24).

[15] L. Silva, J. M. Fernandes, and I. Oliveira, "Limbus: A lightweight remote monitoring system powered by iOS and BLE", ACM, 2015-8. DOI: 10.4108/eai.22-7-2015.2260249.

[16] *Nokia's wibree and the wireless zoo*. [Online]. Available: http://thefutureofthings.com/3041-nokias-wibree-and-the-wireless-zoo/ (visited on 2015-10-24).

[17] *Nokia introduces wibree technology as open industry initiative*. [Online]. Available: http://company.nokia.com/en/news/press-releases/2006/10/03/nokia-introduces-wibree-technology-as-open-industry-initiative.

[18] *Wibree forum merges with bluetooth sig*. [Online]. Available: http://company.nokia.com/en/news/press-releases/2007/06/12/wibree-forum-merges-with-bluetooth-sig (visited on 2015-10-25).

[19] *Bluetooth developer portal*. [Online]. Available: https://developer.bluetooth.org (visited on 2015-10-25).

[20] *Adopted bluetooth profiles, services, protocols and transports*. [Online]. Available: https://www.bluetooth.org/en-us/specification/adopted-specifications (visited on 2015-10-20).

[21] *Fitbit*. [Online]. Available: http://www.fitbit.com (visited on 2015-10-26).

[22] *Wahoo fitness*. [Online]. Available: https://www.wahoofitness.com (visited on 2015-10-20).

[23] H. Strey, P. Richman, R. Rozensky, S. Smith, and L. Endee, "Bluetooth low energy technologies for applications in health care: proximity and physiological signals monitors", *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*, pp. 1–4, 2013. DOI: 10.1109/CEWIT.2013.6851347. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6851347.

[24] J.-r. Lin, T. Talty, and O. Tonguz, "On the potential of bluetooth low energy technology for vehicular applications", *IEEE Communications Magazine*, vol. 53, no. 1, pp. 267–275, 2015, ISSN: 0163-6804. DOI: 10.1109/MCOM.2015.7010544. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7010544.

[25] *Apple unveils iphone*. [Online]. Available: http://www.macworld.com/article/1054769/iphone.html.

[26] Apple, "iOS technology overview", *Media*, p. 69, 2010. [Online]. Available: http://developer.apple.com/library/ios/%7B%5C#%7DDOCUMENTATION/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html.

[27] *Apple introduces the new iphone 3g*. [Online]. Available: http://www.apple.com/pr/library/2008/06/09Apple-Introduces-the-New-iPhone-3G.html (visited on 2015-10-24).

[28] *iOS sdk*. [Online]. Available: https://developer.apple.com/ios/.

[29] *Xcode*. [Online]. Available: https://developer.apple.com/xcode/.

[30] *Xcode features*. [Online]. Available: https://developer.apple.com/xcode/features/.

[31] *Apple developer program — membership details*. [Online]. Available: https://developer.apple.com/programs/whats-included/ (visited on 2015-10-25).

[32] *Apple frees casual iOS developers of membership requirement*. [Online]. Available: http://www.pcworld.com/article/2933052/apple-frees-casual-ios-developers-of-membership-requirement.html (visited on 2015-10-25).

[33] *Apple developer — choosing a membership*. [Online]. Available: https://developer.apple.com/support/compare-memberships/ (visited on 2015-10-25).

[34] *History of bluetooth*. [Online]. Available: http://www.bluetooth.com/Pages/History-of-Bluetooth.aspx (visited on 2015-10-24).

[35] *Mfi program faq*. [Online]. Available: https://mfi.apple.com/MFiWeb/getFAQ.action%7B%5C#%7D1-1 (visited on 2015-10-24).

[36] *Mfi program*. [Online]. Available: https://developer.apple.com/programs/mfi/ (visited on 2015-10-24).

[37] *Corebluetooth framework reference*. [Online]. Available: https://developer.apple.com/library/ios/documentation/CoreBluetooth/Reference/CoreBluetooth%7B%5C_%7DFramework/ (visited on 2015-10-24).

[38] *iOS 5 release notes*. [Online]. Available: https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS5.html (visited on 2015-10-17).

[39] *iOS 6 - release notes*. [Online]. Available: https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS6.html (visited on 2015-10-17).

[40] *Core bluetooth programming guide: objects on the peripheral side*. [Online]. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth%7B%5C_%7Dconcepts/CoreBluetoothOverview/CoreBluetoothOverview.html%7B%5C#%7D//apple%7B%5C_%7Dref/doc/uid/TP40013257-CH2-SW12.

[41] *Core bluetooth programming guide: core bluetooth overview*. [Online]. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothOverview/CoreBluetoothOverview.html#//apple_ref/doc/uid/TP40013257-CH2-SW1 (visited on 2015-10-24).

[42] J. Decuir, "Introducing bluetooth smart: part ii: applications and updates.", *IEEE Consumer Electronics Magazine*, vol. 3, no. 2, pp. 25–29, 2014, ISSN: 2162-2248. DOI: 10.1109/MCE.2013.2297617. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6776529.

[43] *Automatic*. [Online]. Available: https://www.automatic.com (visited on 2015-10-22).

[44] *iOS keys - information property list key reference*. [Online]. Available: https://developer.apple.com/library/prerelease/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/iPhoneOSKeys.html (visited on 2015-10-17).

[45] *Core bluetooth programming guide*. [Online]. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth%7B%5C_%7Dconcepts/AboutCoreBluetooth/Introduction.html (visited on 2015-10-24).

[46] T. Magalhães, I. Oliveira, and J. Fernandes, "Message based integration in cyber-physical system: Firefighters in the field", ACM, 2015-8. DOI: 10.4108/eai.22-7-2015.2260246.

[47] *Xep-0060: publish-subscribe*. [Online]. Available: http://xmpp.org/extensions/xep-0060.html.

[48] J. O'Hara, "Toward a commodity enterprise middleware", *Queue*, vol. 5, no. 4, pp. 48–55, 2007, ISSN: 15427730. DOI: 10.1145/1255421.1255424.

[49] "AMQP specification v1.0 07", *ReVision*, vol. 2011, no. revision 0, 2011.

[50] O. M. Group, "Data distribution service", no. April, 2014. [Online]. Available: http://portals.omg.org/dds/omg-dds-standard/.

[51] *Stomp protocol specification, version 1.2*. [Online]. Available: http://stomp.github.io/stomp-specification-1.2.html.

[52] *Extensible messaging and presence protocol (xmpp): core — rfc6120*. [Online]. Available: https://tools.ietf.org/html/rfc6120.

[53] *Mqtt version 3.1.1*. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[54] *Mqttkit*. [Online]. Available: https://github.com/jmesnil/MQTTKit (visited on 2015-10-23).

[55] *Cooking hacks*. [Online]. Available: https://www.cooking-hacks.com (visited on 2015-10-24).

[56] *Vital responder: monitoring stress and fatigue in first responders*. [Online]. Available: http://wiki.ieeta.pt/wiki/index.php/VitalResponder (visited on 2015-10-17).

[57] *Mbed nordic nrf51-dk*. [Online]. Available: https://developer.mbed.org/platforms/Nordic-nRF51-DK/ (visited on 2015-10-13).

[58] *Core bluetooth programming guide: performing common central role tasks*. [Online]. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth%7B%5C_%7Dconcepts/PerformingCommonCentralRoleTasks/PerformingCommonCentralRoleTasks.html%7B%5C#%7D//apple%7B%5C_%7Dref/doc/uid/TP40013257-CH3-SW1 (visited on 2015-10-26).

[59] *Mosquitto - an open source mqtt v3.1/v3.1.1 broker*. [Online]. Available: http://mosquitto.org (visited on 2015-10-26).

[60] *Mosquitto - configuration file*. [Online]. Available: http://mosquitto.org/man/mosquitto-conf-5.html (visited on 2015-10-26).

[61] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)", [Online]. Available: http://tools.ietf.org/html/rfc7252.

[62] C. Bormann, A. P. Castellani, Z. Shelby, U. Bremen, and Z. S. Sensinode, "Coap: an application protocol for billions of tiny internet nodes", *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012, ISSN: 10897801. DOI: 10.1109/MIC.2012.29.

[63] *Icoap*. [Online]. Available: https://github.com/stuffrabbit/iCoAP (visited on 2015-10-23).

[64] *Nsnotificationcenter class reference*. [Online]. Available: https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/Foundation/Classes/NSNotificationCenter%7B%5C_%7DClass/ (visited on 2015-10-26).

[65] *Nsuserdefaults class reference*. [Online]. Available: https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSUserDefaults%7B%5C_%7DClass/index.html%7B%5C#%7D//apple%7B%5C_%7Dref/occ/cA.l/NSUserDefaults (visited on 2015-10-26).

[66] *Uidevice class reference*. [Online]. Available: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDevice%7B%5C_%7DClass/index.html%7B%5C#%7D//apple%7B%5C_%7Dref/occ/instp/UIDevice (visited on 2015-10-25).

[67] *Django*. [Online]. Available: https://www.djangoproject.com (visited on 2015-10-25).

[68] *Cfnetwork error codes reference*. [Online]. Available: https://developer.apple.com/library/mac/documentation/Networking/Reference/CFNetworkErrors/index.html%7B%5C#%7D//apple%7B%5C_%7Dref/c/tdef/CFNetworkErrors (visited on 2015-11-20).